JavaScript Fundamentals:

Instructor Guide

CIW Web Languages Series and CIW Web Developer Series

CCN02-CEJSRF-PR-210 • version 5.1 • rd092602

CIW Certification

The CIW Internet skills certification program, sponsored by the National Workforce Center for Emerging Technologies (NWCET), is aimed at professionals who design, develop, administer, secure and support Internet- or intranet-related services. It offers a unique opportunity to learn, demonstrate and prove competence on Web-related technologies. CIW certifications are endorsed by the International Webmasters Association (IWA) and the Association of Internet Professionals (AIP).

Benefits of Certification

CIW Internet skills certification offers industry-wide recognition of an individual's Internet and Web knowledge and skills. It also provides tangible evidence of a person's competency as an Internet professional and is frequently a factor in hiring and assignment decisions. Individuals holding CIW certification can demonstrate to potential employers and clients that they have passed rigorous training and examination requirements, setting them apart from non-certified competitors. The CIW logo identifies individuals as Internet professionals who have been certified by one of the most prestigious programs in the industry.

Certification Testing

CIW certification exams are administered by Prometric, Inc. through Authorized Prometric Testing Centers (APTCs) and by Virtual University Enterprises (VUE) testing centers worldwide. The exams measure technical proficiency and establish a level of core competency required to become CIW certified. To register for a CIW exam online, visit Prometric at *www.2test.com* or VUE at *www.vue.com*. For more information about CIW exams, visit *www.CIWcertified.com* and select the Exams link.

We encourage you to speak with the CIW training partner of your choice to find out more about the vendor-neutral CIW Internet Skills program. To locate a CIW training partner, visit www.CIWcertified.com, select the Training Locator link and select the training partner of your choice, either by location, partner or course.

www.CIWcertified.com

CIW and NWCET…

Setting IT Skills Standards for the Knowledge Economy

The National Workforce Center for Emerging Technologies (NWCET) is the official sponsor of the CIW job-role certification program and leader of the CIW Certification Council. NWCET's involvement ensures the promotion of the CIW credential as a practical way to implement skill standards and assures that the CIW program continues to validate the skills required to succeed in the competitive technology-enabled workforce.

NWCET developed the only nationally recognized and validated IT skills standards, funded in part by the National Science Foundation, which comprehensively describe the skills, knowledge and attributes required of professionals in the knowledge economy.

NWCET's leadership role in CIW is part of its ongoing mission to lead education, business and government in developing skilled information technology (IT) workers.

Please visit *www.nwcet.org* for more information regarding NWCET, CIW or the CIW Certification Council.

CIW and IWA…

Promoting a Worldwide Standard for Web Certification!

IWA, the non-profit association for Web professionals, is the industry's recognized leader in providing educational and certification standards. IWA initiatives support more than 100 official chapters with 14,000 individual members in 106 countries.

IWA has integrated industry-leading CIW Internet skills training into its Certified Web Professional program to offer a single worldwide standard in Web certification. To be eligible for IWA Certified Web Professional status, a candidate must pass CIW Foundations and a CIW job role series exam, demonstrate a minimum of two years' related experience, and complete a job role series every three years by completing thirty hours of continuing education.

MEMBER
CIW FOUNDATIONS
CIW PROFESSIONAL
CWP CERTIFIED
MEMBER

IWA Certified Web Professional (CWP) status provides Web certification based on an individual's proven experience, professionalism and knowledge. Please visit us at *www.iwanet.org* for more information about applying your CIW credential for CWP status.

JavaScript Fundamentals:
Instructor Guide

prosofttraining™

Customer Service
ComputerPREP
A division of ProsoftTraining
410 N. 44th Street, Suite 600
Phoenix, AZ 85008
(602) 275-7700

JavaScript Fundamentals

Developer

Brian Danks

Contributor

Lisa Pease

Editor

Susan Lane

Publisher

Joseph A. Servia

Project Managers

Dave De Ponte and Todd Hopkins

ISBN: 1-58143-844-3

Table of Contents

List of Labs

List of Optional Labs

List of Quizzes

List of Figures

List of Tables

Course Description

Welcome to *JavaScript Fundamentals,* a 12-hour course designed to teach you the features of the JavaScript language. This course will empower you with the skills to design client-side, platform-independent solutions that greatly increase the value of your Web site. The first part of this course teaches you foundational JavaScript skills. The second part will build on your knowledge and present solutions for more functional and exciting Web pages. You will learn how to communicate with users, script for the JavaScript object model, control program flow, validate forms, animate images, target frames, and create cookies. By the end of this course, you will understand and use the most popular applications of JavaScript.

Length

JavaScript Fundamentals is a 12-hour course.

Series

JavaScript Fundamentals is the first course in the CIW Web Languages series. CIW Web Languages consists of the following 2 courses:

- *JavaScript Fundamentals*
- Perl Fundamentals

The two parts of the CIW Web Languages series do not build upon one another. Rather, each course provides a complete introduction and development in the discipline of languages used on the Web. The first course discusses the fundamentals of the JavaScript scripting language, whereas the second course focuses on the fundamentals of the Perl programming language.

The *JavaScript Fundamentals* course is also part of the CIW Web Developer Series. For more information, please visit the *www.ciwcertified.com* Web site.

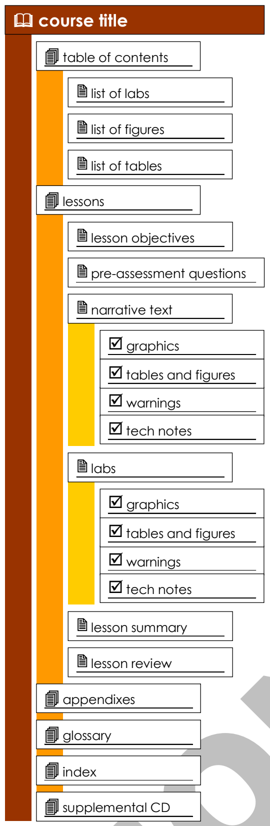
Prerequisites

Students must have completed the *CIW Foundations Series* or be able to demonstrate equivalent Internet knowledge.

ProsoftTraining Courseware

This coursebook was developed for instructor-led training and will assist you during class. Along with comprehensive instructional text and objectives checklists, this coursebook provides easy-to-follow hands-on labs and a glossary of course-specific terms. It also provides Internet addresses needed to complete some labs, although due to the constantly changing nature of the Internet, some addresses may no longer be valid.

The student coursebook is organized in the following manner:



When you return to your home or office, you will find this coursebook to be a valuable resource for reviewing labs and applying the skills you have learned. Each lesson concludes with questions that review the material. Lesson review questions are provided as a study resource only and in no way guarantee a passing score on CIW exams.

The course is available as either an academic or a learning center/corporate version. Each of these versions has an instructor book and student books. Check your book to verify which version you have, and whether it is an instructor or student book. Following is a brief discussion of each version.

- **Academic:** Designed for students in an academic classroom environment; typically taught over a quarter (10-week) or semester (16-week) time period. Example syllabi are included on the instructor CD-ROM. The instructor's book and CD-ROM contain all answers, as well as activities (pen-and-paper-based labs), optional labs (computer-based labs), quizzes, a course assessment, and the accompanying handouts for the instructor to assign during class or as homework. No answers exist in the student book or on the student CD-ROM. Students will have to obtain answers from the instructor.
- **Learning Center or Corporate:** Designed for students in a learning center/corporate classroom environment; typically taught over a one- to five-day time period (depending on the length of the course). An example implementation table is included on the instructor CD-ROM. Similar to the academic version, the instructor's book and CD-ROM contain all answers, as well as activities (pen-and-paper-based labs), optional labs (computer-based labs), quizzes, a course assessment, and the accompanying handouts for the instructor to assign during class or as homework. However, the student CD-ROM contains answers, including those to the pre-assessment questions, labs, review questions, activities, optional labs, quizzes and course assessment.

Course Objectives

After completing this class, you will be able to:

- Describe the origins of JavaScript and list its key characteristics.
- Communicate with users using JavaScript.
- Define and call JavaScript functions.
- Control program flow.
- Explain and use the JavaScript object model.
- Identify and use the JavaScript language objects.
- Use JavaScript with HTML form controls.
- Define and use cookies.
- Discuss security issues relevant to JavaScript.
- Create custom JavaScript objects.

Classroom Setup

Your instructor has probably set up the classroom computers based on the system requirements listed below. Most software configurations on your computer are identical to those on your instructor's computer. However, your instructor may use additional software to demonstrate network interaction or related technologies.

System Requirements

Hardware

The following table summarizes the hardware requirements for all courses in the CIW program. Each classroom should be equipped with enough personal computers to accommodate each student and the instructor with his or her own system.

Note: The CIW hardware requirements are similar to the lowest system requirements for Microsoft implementation (Level 1 requirements) except that CIW requires increased hard disk space (8 GB) and RAM (128 MB). This comparison may be helpful for the many training centers that implement CIW and are also CTEC because personnel at these centers are familiar with the Microsoft hardware specifications.

CIW hardware specifications	Greater than or equal to the following
Processor	Intel Pentium II (or equivalent) personal computer with processor speed greater than or equal to 300 MHz
L2 cache	256 KB
Hard disk	8-GB hard drive
RAM	At least 128 MB
CD-ROM	32X
Network interface card (NIC)	10BaseT or 100BaseTX (10 or 100 Mbps)
Sound card/speakers	Required for instructor's station, optional for student stations
Video adapter	At least 4 MB
Monitor	15-inch monitor
Network hubs	Two 10-port 10BaseT or 100BaseTX (10 or 100 Mbps) hubs
Router	Multi-homed system with three NICs (Windows NT 4.0/2000 server)*

** Must meet universal CIW hardware requirements.*

Software

The recommended software configurations for computers used to complete the labs in this book are as follows.

To be installed before class:

- Microsoft Windows Millennium Edition (Me) or Windows 2000.
- Internet Explorer 5.5 (or later) or Netscape Navigator 4.0 (or later).
- A standard text editor (usually available with the operating system).

Connectivity

Internet connectivity is required for this course. The minimum requirement is a modem and an Internet service Provider (ISP) account. However, you will experience improved results with a 56-Kbps modem with a Point-to-Point Protocol (PPP) account through an ISP, or a direct Internet connection.

Conventions and Graphics Used in This Book

The following conventions are used in Prosoft coursebooks.

Terms	Technology terms defined in the margins are indicated in **bold** the first time they appear in the text. Not every word in bold is a term requiring definition.
Lab Text	Text that you enter in a lab appears in **bold**. Names of components that you access or change in a lab also appear in **bold**.
Notations	*Notations or comments regarding screenshots, labs or other text are indicated in italic type.*
Program Code or Commands	Text used in program code or operating system commands appears in the `Lucida Sans Typewriter` font.

The following graphics are used in Prosoft coursebooks.

Tech Notes point out exceptions or special circumstances that you may find when working with a particular procedure. Tech Notes that occur within a lab are displayed without the graphic.

Tech Tips offer special-interest information about the current subject.

Warnings alert you about cautions to observe or actions to avoid.

This graphic signals the start of a lab or other hands-on activity.

Each lesson summary includes an *Application Project*. This project is designed to provoke interest and apply the skills taught in the lesson to your daily activities.

Each lesson concludes with a summary of the skills and objectives taught in that lesson. You can use the Skills Review checklist to evaluate what you have learned.

↘ This graphic indicates a line of code that is completed on the following line.

Classroom Setup Guide

The *JavaScript Fundamentals* v5.1 Classroom Setup Guide is divided into three sections:

1. **Before You Begin** includes courseware update links for instructors, a revision history outlining the revisions made to a coursebook since the last version, an explanation of the requirements for preparing a classroom behind a proxy server, and additional notes that you should consider before you set up the classroom.
2. **Classroom Requirements** lists the hardware, software and connectivity requirements to implement this course.
3. **Setup Instructions** include the configuration requirements for both instructor and student systems, and a detailed list of required software installation procedures.

Before You Begin

This section includes courseware update links for instructors, a revision history outlining the revisions made to a coursebook since the last version, an explanation of the requirements for preparing a classroom behind a proxy server, and additional notes that you should consider before you set up the classroom.

Courseware updates

Instructors must download the latest courseware updates from the Instructor Community on the CIW Web site (*www.ciwcertified.com*) before teaching the course. CIW courseware is updated continually, and the courseware updates provide the most current changes, revisions and notes for all CIW courseware.

Courseware updates include feedback from ATPs, clients and instructors who implement the CIW program. Feedback is reviewed and updates are posted in dynamic documents for both students and instructors. Each document correlates with the identical version of the coursebook (e.g., *v4.1 Update* is designed to be used only with v4.1 of the coursebook). Updates are available for both the current version and the version immediately previous to the current version of the coursebooks. CIW does not provide support for coursebooks and instructor materials that are two or more versions removed from the current versions.

Revision history

Released October 2002 (version 5.1)

This release is considered a course enhancement. The main differences between JavaScript Fundamentals v5.1 and the previous version (v5.07 released March 2002) are as follows:

- Applied new publishing template to course for more professional look and feel
- Entered errata (an error in writing or printing, including but not limited to spelling, style and code errors)

Released March 2002 (version 5.07)

This release was considered a course enhancement. The main differences between JavaScript Fundamentals v5.07 and the previous version (v4.07 released August, 2001) were as follows:

- Introduction of separate instructor and student guides.
- Pre-assessment questions.
- Instructor margin notes.
- Enhanced Lesson Summary.
- Instructor Section containing Optional Labs and quizzes.
- Syllabi (Academic) and implementation table (Learning Center).
- All labs are streamlined to use individual files for each lab. The precreated Web site that was used in earlier versions of this course is no longer being used.

Released August 2001 (version 4.07)

Several significant changes were made to the JavaScript Fundamentals courseware, outlines or objectives from the April 2001 release. The main differences between this course and the earlier version of JavaScript Fundamentals (v3.17), released April 1, 2001, were as follows:

- Errata updates were entered (any errors in writing or printing, including but not limited to spelling, style and code errors). Most labs were redesigned to challenge the students to create code without copying JavaScript from the coursebook. Students were shown a concept, and then were asked to demonstrate that concept in the labs. As always, completed files were included for students who prefer to copy code or want to execute the labs without typing them.
- Most labs were redesigned to minimize the amount of typing the students must perform, allowing more time to focus on JavaScript concepts with less time focused on tracing typographical errors.
- Explanations of labs and example code were expanded throughout the coursebook.
- Most HTML content was removed from the main coursebook and placed in an appendix, including content that discussed the creation of HTML image maps and the targeting of HTML frames. The content was included in an appendix if needed, but was removed from the JavaScript content in order to streamline the course.
- Lesson 10 from v3.17 was removed and its JavaScript content was placed elsewhere in the coursebook.
- Lesson 11 (Custom JavaScript Objects) became Lesson 10. Discussion of complex JavaScript custom objects was removed from this lesson and placed in an appendix.
- Several helpful discussions were added to the appendix. These topics include JavaScript operator precedence, the concepts of passing function arguments by value and by reference, the sort() method of the Array object, complex custom objects and the aforementioned HTML content.

Preparing the classroom behind a proxy server

Suggestion

If Internet access is required (or preferred) for a class and the classroom is behind a proxy server, you may have problems downloading programs during classroom setup and completing certain labs during class. Most proxy servers already allow HTTP traffic; however, difficulties may arise when you require additional services, such as e-mail, FTP or program downloads.

Two suggestions are offered:

1. Talk with the network administrator at the location and make sure that:

    a. The classroom has proper access to all Internet-related protocols used in the class. Examples include HTTP (TCP/UDP port 80), SSL (TCP/UDP port 443), FTP (TCP/UDP port 20, 21), Telnet (TCP/UDP port 23), POP3 (TCP/UDP port 110), SMTP (TCP/UDP port 25), NNTP (TCP/UDP port 119) and Gopher (TCP/UDP port 70). For certain services, such as FTP, you will need all ports above 1023 (registered ports).

    b. The IP addresses assigned to the computers in your classroom have permission to access the Internet.

2. Download all the required software (with proper licensing) for the course before you arrive at the site, and place the source files on the instructor's computer. Students can then access all source files from shares that you create. Perhaps an instructor can create a CD-ROM with the required software source files. This will not solve the issues addressed in the first suggestion, but will solve any problems concerning downloads.

Classroom Requirements

This section lists the hardware, software and connectivity requirements to implement this course.

Hardware requirements

The following table summarizes the hardware requirements for all courses in the CIW program. Each classroom should be equipped with one instructor station and x number of student stations (for example, in a classroom with 13 personal computers, set one up as the instructor station and the remaining 12 as student stations).

The CIW hardware requirements are similar to the lowest system requirements for Microsoft implementation (Level 1 requirements) except that CIW requires increased hard disk space (8 GB) and RAM (128 MB). This comparison may be helpful for the many training centers that implement CIW and are also CTEC because these centers are familiar with the Microsoft hardware specifications.

CIW hardware specifications	Greater or equal to the following
Processor	Intel Pentium II or equivalent, personal computer with processor speed greater than, or equal to 300 MHz.
L2 cache	256 KB
Hard disk	8-GB hard drive
RAM	At least 128 MB
CD-ROM	32X
Network Interface Card (NIC)	10BaseT or 100BaseTX (10 or 100 Mbps)
Sound card/speakers	Required for instructor's station, optional for student stations
Video adapter	At least 4 MB
Monitor	15-inch monitor
Network hubs	Two 10-port 10BaseT or 100BaseTX (10 or 100 Mbps hubs)
Router	Multi-homed system with three NICs (Windows NT 4.0/2000 server)*

** Must meet universal CIW hardware requirements.*

Software requirements

Install the following software on the instructor and student systems. The instructor and student configurations are identical.

- Microsoft Windows Millennium Edition (Windows 2000 can also be used)
- Internet Explorer 5.5 or later (Netscape Navigator 4.0 or later can also be used).
- A standard text editor, such as Notepad or WordPad (included with Microsoft Windows).

Connectivity requirements

This class requires Internet access. Below are two options available to you.

1. Obtain valid IP addresses from a DHCP server.
2. If no DHCP server is available, obtain TCP/IP configurations from the network administrator, and manually configure each system.

Setup Instructions

Use the following procedures to set up the computers for class. You will install exactly the same software on both the instructor and the student systems.

To set up the hardware

Set up the hardware according to the manufacturer's instructions. (Refer to the hardware requirements.)

To set up the software

To install and configure Windows Millennium Edition:

1. Install Microsoft Windows Millennium Edition (standard installation) with the following parameters.

When this information is requested	Use
Installation Directory	C:\Windows (recommended)
Setup Options	Typical
Windows Components	Install the most common components (recommended)
Computer Name	Instructor – instructor computer Student*X* - *X* is the assigned number for the student. For instance, if 20 students are in the class, then name the computers student1, student2, student3, etc., until you reach student20.
Workgroup name	Classroom
Startup Disk	Optional
Windows Password	Optional

Note: If you are teaching where several classrooms are connected, you may encounter name conflicts. If so, add a number to each name. For example, name the instructor's computer Instructor1 and the workgroup Classroom1.

2. DHCP is used by default with a Typical installation of Millenium Edition. Therefore, no network configurations are necessary. If your network does not have a DHCP server, obtain a valid IP address, subnet mask, default gateway and DNS server information from your network administrator.

To install and configure Microsoft Internet Explorer 5.5

- Microsoft Internet Explorer 5.5 with Outlook Express is installed during the Windows Millenium Edition **Typical** setup. Therefore, no installation instructions are required.

Supplemental CD-ROM

Each coursebook includes a supplemental CD-ROM. The files on the CD-ROM are important, as they are referenced and used throughout the course. Because the labs will explicitly refer to the CD-ROM, the instructor and students will access and load these files during class. The instructor will find many additional tools on the CD-ROM, including a slide show, and electronic versions of the handouts and answer files.

Lesson 1: Introduction to JavaScript

Objectives

By the end of this lesson, you will be able to:

- Describe the origins of JavaScript.
- List the key JavaScript characteristics.
- Describe the differences between Java and JavaScript.
- Discern among JavaScript, JScript and VBScript.
- Differentiate among server-side and client-side JavaScript applications.
- Embed JavaScript into HTML.
- Use the JavaScript comment tags.

Pre-Assessment Questions

1. Which of the following tools can run client-side JavaScript?

    a. *Microsoft Internet Explorer 3.0 or later*
    b. Netscape Navigator 1.0 or later
    c. *Netscape Navigator 2.0 or later*
    d. Microsoft Internet Explorer 2.0 or later

2. Which of the following statements properly references an external JavaScript file?

    a. `<SCRIPT TYPE="text/javascript" script_src="myJSCode.js">`
    b. `<SCRIPT TYPE="text/javascript" source="myJSCode.js">`
    c. *`<SCRIPT TYPE="text/javascript" src="myJSCode.js">`*
    d. `<SCRIPT TYPE="text/javascript" script_source="myJSCode.js">`

3. Briefly describe the difference between an event-driven programming model and a procedural programming model.

    In the event-driven programming model, programs are written to respond to user-generated events and actions. Programming modules, such as subroutines or functions, are created as independent entities and perform their tasks when the appropriate action (or event) takes place. The procedural programming model executes a program sequentially and usually in a predetermined order. The user has much less control of the manner in which a procedural program will execute.

Introduction to Scripting

INSTRUCTOR NOTE: Netscape Corporation started developing LiveScript in late 1995. In early December 1995, just before Netscape Navigator 2.0 was released, Netscape and Sun announced that LiveScript would be known as JavaScript. This name change was done more for marketing purposes than anything else. Much confusion was initially caused by the similarities of the names Java and JavaScript. The differences between the two are discussed later in this lesson.

When the World Wide Web first became popular, Hypertext Markup Language (HTML) was the only language an author could use to create Web pages. HTML is not a programming language but a "markup" language and has many limitations. HTML positions text and graphics on a Web page but offers limited interactivity within the Web page. Most computer users, whether they use Windows, Macintosh, UNIX or some combination, are now accustomed to graphical application interfaces. They click buttons to execute command sequences, enter values into text boxes, and choose from menu lists. This increase in user abilities and expectations has resulted in a continual improvement of HTML, as well as the advent of powerful scripting languages such as JavaScript.

Origins of JavaScript

Netscape Corporation developed the JavaScript language. JavaScript is not a stand-alone programming language like Java. When JavaScript was first developed, its name was LiveScript, but Netscape contracted with Sun Microsystems to name it JavaScript, more due to Java's popularity than any similarity between the two languages.

To run properly, client-side JavaScript is written within HTML documents. Server-side JavaScript, called LiveWire, can work with back-end server processes. Whether used for client-side or server-side solutions, JavaScript allows programmers to add interactivity to Web pages without using server-based applications, such as Common Gateway Interface (CGI) programs.

JavaScript was first supported in Navigator version 2.0, and has since gained universal support in such browsers as Internet Explorer 3.0 and later, Mosaic 2.0 and later, and Lotus Personal Web Client 3.0 and later.

JavaScript Characteristics

Before you start writing in JavaScript, you should take a close look at the features of this language.

JavaScript is a scripting language

A scripting language is a simple programming language designed to enable computer users to write useful programs easily. Scripting languages, including **Perl**, **Tcl** and **REXX**, are interpreted, meaning that they are not compiled to any particular machine or operating system. This feature makes them platform-independent.

Perl
A script programming language commonly used for Web server tasks and CGI programs.

Tool Command Language (Tcl)
An interpreted script language used to develop applications such as GUIs, prototypes and CGI scripts.

REXX
A procedural programming language used to create programs and algorithms.

One of the key uses for scripting languages such as JavaScript is to allow more complex programs, created by programming languages such as C and C++, to work together. JavaScript is one of the more popular languages, and is uniquely suited for this purpose.

If you have ever written a macro in Microsoft Excel or used WordBasic to perform some task in a Microsoft Word document, you have already used a scripting language. Smaller and less powerful than full programming languages, scripting languages provide easy functionality. JavaScript syntax is similar to that of C or Pascal.

JavaScript is object-based, not object-oriented

Object-oriented is a common term in programming languages. An object-oriented program is a collection of individual objects that perform different functions, rather than a sequence of statements that collectively perform a specific task. These objects are usually related in a hierarchical manner, in which new objects and subclasses of objects inherit the properties and methods of the objects above them in the hierarchy. JavaScript is not object-oriented because it does not allow for object inheritance and subclassing in the traditional sense. However, JavaScript is an *object-based* language because it derives functionality from a collection of built-in objects. With JavaScript, you can also create your own objects. The concept of objects will be discussed further later in this lesson.

JavaScript is event-driven

The World Wide Web is based upon an event-driven model. For example, whenever you click an item on a Web page, an event occurs. The previous programming model was the procedural model, in which the user (if there is one) is expected to interact with the program in a fairly sequential manner. On a Web page, however, the user is in control and can click or not click, move the mouse or not move the mouse, or change the URL at will. Because of the unpredictability of a user's actions, programming modules (called subroutines or functions) can be created that are independent of each other and do not require any sequential set of operations.

Events can trigger functions. Event triggers can be as simple as the user clicking a button, clicking or moving the mouse over a hyperlink, or entering text into a text field. Scripting can be tied to any of these events. You will learn to use JavaScript to instruct the browser what to do or display using the event that you designate as the trigger for the script.

Client-side JavaScript is part of the text within your HTML document. When your browser retrieves a scripted page, it executes the JavaScript programs and performs the appropriate operations in response to user events.

JavaScript is platform-independent

user agent
The W3C term for any application, such as a Web browser or help engine, that renders HTML for display to users.

Because JavaScript programs are designed to run within HTML documents, they are not tied to any specific hardware platform or operating system. However, JavaScript programs are tied to a specific **user agent**. Generally, these user agents are browsers. Theoretically, you can implement the same JavaScript program on any current user agent, such as Netscape Navigator 2.0 or later, or Internet Explorer 3.0 or later.

INSTRUCTOR NOTE: A demanding element of creating JavaScript programs for the World Wide Web is keeping up to date with JavaScript versions and the browsers that support them. This fact should be stressed early and often to new JavaScript programmers.

Keep in mind that each user agent tends to implement JavaScript differently. Different vendors and user agent versions can complicate your JavaScript implementation. Because you usually cannot guarantee that a user will access your JavaScript code using a specific user agent, take care to create code that will run on as many platforms as possible. JavaScript provides a way for you to determine the user agent used to access your programs.

JavaScript enables quick development

Because JavaScript does not require time-consuming compilation, scripts can be developed quickly. This advantage is enhanced by the fact that most of the interface features, such as forms, frames and other **graphical user interface (GUI)** elements, are handled by the browser and HTML code. JavaScript programmers need not worry about creating or handling these elements of their applications.

graphical user interface (GUI)
A program that provides graphical navigation with menus and screen icons.

JavaScript is relatively easy to learn

JavaScript does not have the complex syntax and rules associated with Java. Even if you do not know any other programming language, learning JavaScript will not be difficult.

JavaScript and Common Programming Concepts

Some key JavaScript concepts may not make sense right away. However, as this course examines various examples and labs, you will acquire a better understanding of these concepts.

Scripting languages are subsets of larger, more complicated languages. They provide less functionality than full programming languages, but are usually easier to learn. Your investment in learning a scripting language is valuable if you decide to learn a full programming language because you will gain basic conceptual awareness of common programming practices.

Objects, properties and methods

Serious programmers tend to write code in C++, Visual Basic or Java. These three high-level languages provide rich functionality to the program developer. They are also "object-oriented" languages. In programming, **objects** encapsulate predesignated attributes and behavior. They are often grouped with similar objects into classes.

object
A programming function that models the characteristics of abstract or real "objects" using classes.

Like real-life objects, JavaScript objects have certain attributes and behaviors. Developers refer to attributes and behaviors with three other terms: **properties**, **values** and **methods**. Properties represent various attributes of an object, such as height, color, font size, sentence length and so forth. Values represent the specific qualities of properties. For instance, the statement `color="red"` assigns a value to a property. Methods are the actions that an object can be made to perform, such as a calculation, an onscreen move or the writing of text in a window. Methods often describe the actions that an object performs with its properties.

property
A descriptive characteristic of an object, such as color, width or height, that the programmer stipulates in the creation of the object.

To further explain objects, properties, values and methods, consider a pen in terms of object-based programming. It is clearly an object. A pen has definite, discernable properties, such as a length, ink color, point style and so forth. Two pens may have similar properties, yet they may have different values for those properties: All pens will have a color (i.e., property), but all pens will not have the same color (i.e., value).

value
The specific quality such as color, width or height that belongs to the property of an object.

A pen has methods as well. It can write, spin, flip, and have its cap removed or replaced. A developer creating a virtual pen object would simulate natural attributes and behavior so that, for example, if users try to write with pens, they might receive errors directing them to first remove the caps.

method
An action that can be performed by an object.

INSTRUCTOR NOTE:
If needed, refer to **Instructor Note 1-1** for another example of objects, methods and properties.

A sentence on a page is another example of an object. You can think of a sentence as a string of words. But to a programmer, that sentence might be an object that has minimum properties of characters, color and font. The values would include number of characters (or sentence length), color type and font style. In addition, you may want to invoke a method in which you can convert that sentence to all uppercase or all lowercase letters.

Ultimately, objects can be defined as collections of properties, with collections of methods that operate on those properties. As you work and develop an understanding of JavaScript, you will see many examples of objects along with their properties and methods.

Java and JavaScript

Although the names are similar, Java and JavaScript are different languages. Java is a full-fledged object-oriented programming language. Developed by Sun Microsystems, Java can be used to create stand-alone applications and a special type of mini-application called a Java applet. Applets are written in Java, compiled, and then referenced via the `<APPLET>` tag (or `<OBJECT>` in HTML 4.0) in a Web page. Applets can provide a great variety of added functionality to Web sites.

INSTRUCTOR NOTE: Stress the differences between Java and JavaScript to new JavaScript programmers. Many students may have been exposed to many of the misconceptions caused by the similar names of the two languages.

JavaScript is an object-based scripting language. Although it uses some of Java's expression syntax and basic program flow controls, JavaScript stands alone and does not require Java. In fact, any similarities between the two are due to their use of objects. JavaScript was not developed by the same company, nor was it developed with Java in mind. Figure 1-1 illustrates how the similarities are tangential.

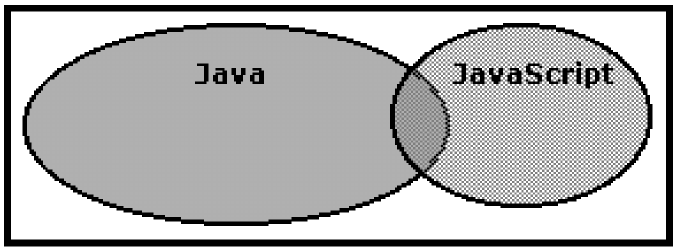


Figure 1-1: Comparing Java to JavaScript

Table 1-1 further compares JavaScript and Java.

Table 1-1: Comparison of JavaScript and Java

JavaScript	Java
Interpreted (not compiled) by client.	Compiled on server before execution on client.
Object-based; code uses built-in, extensible objects, but no classes or inheritance.	Object-oriented; applications and applets consist of object classes with inheritance.
Variable data types not declared (loose typing).	Variable types must be declared (strong typing).
Dynamic binding; object references checked at run time.	Static binding; object references must exist at compile time.
Secure; cannot write to hard disk.	Secure; cannot write to hard disk.
Code integrated with and embedded in HTML.	

JavaScript and VBScript

JavaScript and VBScript are scripting languages that have similar purposes. Both extend the capabilities of static Web pages.

JavaScript was the first scripting language developed for Web page design. It has been incorporated into the popular Netscape Navigator browsers since version 2.0. VBScript is the Microsoft Web-scripting language, based on the powerful and popular Visual Basic language.

JavaScript is a strong object-based language that relies, for much of its functionality, on objects and their attendant methods and properties.

VBScript relies much less upon the traditional object classes and much more on dynamic, built-in customizing functions. Is one scripting language better? It depends entirely on the situation and the programmer's knowledge of that particular language. Programmers must consider that client-side VBScript is only supported by the Microsoft Internet Explorer browser.

Visual Basic and VBScript

Just as JavaScript is a separate language that has similarities to Java, VBScript is a language that has similarities to Visual Basic. In contrast, however, VBScript is a subset of the Visual Basic language.

JavaScript, JScript and ECMAScript

The Microsoft implementation of the Netscape JavaScript language is called JScript. Several differences exist between these two implementations. Although some of the differences are minor, others cause compatibility problems.

Netscape, the inventor of JavaScript, announced in 1997 that the European Computer Manufacturer's Association (ECMA) had approved JavaScript as an international standard. In some circles, JavaScript is now officially known as ECMAScript, and many hope that it will become widely accepted. The ECMA standard intends to diminish the differences between JavaScript and JScript. The following description is quoted from the ECMA Web site:

"This ECMA Standard is based on several originating technologies, the most well known being JavaScript (Netscape) and JScript (Microsoft)."

For more information about ECMAScript, consult the ECMA home page at *www.ecma.ch/*. This site refers to the ECMAScript standard as Standard ECMA-262. The most recent specification is the third edition, published in December 1999.

JavaScript versions

Currently, six versions of JavaScript exist. Table 1-2 lists the various versions of JavaScript and the implementations of Netscape Navigator and Microsoft Internet Explorer that support each version. Corresponding versions of JScript and support environments are also listed.

Table 1-2: JavaScript versions and browser support

Version	Netscape Navigator Support	Microsoft Internet Explorer Support
JavaScript 1.0 JScript 1.0	Navigator 2.x	Internet Explorer 3.x
JScript 2.0	N/A	Microsoft Internet Information Server 1.0
JavaScript 1.1	Navigator 3.x	N/A
JavaScript 1.2 JScript 3.0	Navigator 4.0 through 4.05	Internet Explorer 4.x Microsoft Internet Information Server 4.0 Microsoft Windows Scripting Host 1.0
JavaScript 1.3	Navigator 4.06 through 4.7x	N/A
JScript 4.0	N/A	Microsoft Visual Studio 6.0
JavaScript 1.4 JScript 5.0	Did not map to a release of Navigator	Internet Explorer 5.x
JavaScript 1.5	Navigator 6.0	N/A
JScript 5.5	N/A	Microsoft Internet Information Services 5.0

One of the challenges of creating JavaScript programs is remembering the various features offered by the different versions of the language. You must also be aware of your target audience and the browsers they most likely use. Currently, Netscape Navigator and Microsoft Internet Explorer are the two most popular browsers. Most Web applications are built to target these two browsers.

INSTRUCTOR NOTE:
Refer to instructor section for URLs that will provide information about JavaScript and JScript versions.

Fortunately, the core functionality offered by JavaScript does not change from version to version. As new versions of the language are released, new features are added, but the basic features remain the same.

Server-Side vs. Client-Side Applications

The focus of this course is client-side JavaScript. You can use client-side JavaScript, for example, to validate specific form fields of data to make sure the client has entered the required fields. By using JavaScript wisely, you can begin processing client information before it reaches the more resource-intensive server processes.

JavaScript can also be used in server-side solutions. For instance, after the fields of data are submitted to a server, server-side JavaScript can be used to parse and disseminate the information.

The key to using JavaScript properly is remembering that the wisest uses of JavaScript unify client-side and server-side solutions.

Server-side applications and LiveWire

LiveWire, JavaScript's server-side solution, enables you to connect Web pages to databases, enables server-side image maps and saves client state so the computer will remember where the client is during a multi-page process. LiveWire is an add-on package that works with Netscape Enterprise Server.

LiveWire technology includes server-side programming objects that allow the programmer to detect and respond to information sent from client browsers. These objects also enable the programmer to work with and manipulate relational databases such as Informix, Oracle and Sybase. LiveWire also provides a WYSIWYG (What You See Is What You Get) editor/browser and a graphical Web site manager. JavaScript can be used on either the server side or the client side to work with LiveWire. However, server-side JavaScript will function only if LiveWire is installed.

Currently, JavaScript does not support direct database access without LiveWire.

Client-side applications

This course teaches you how to implement client-side applications. Client-side JavaScript can be used to build many scalable solutions that allow you to send messages to users, launch new windows, work with frames, create your own objects and more.

Embedding JavaScript into HTML

JavaScript resides within an HTML document. Authors usually place it into an HTML document using the <SCRIPT> tag. You can add script to the head or the body section (or both) of an HTML document. In this course, you will learn how to embed scripting instructions directly into certain HTML tags as well, a technique called inline scripting.

The basic structure of an HTML file with JavaScript is as follows (JavaScript indicated in bold):

```
<HTML>
<HEAD>
<TITLE>Page Title</TITLE>

<SCRIPT LANGUAGE="JavaScript">
<!–

//JavaScript code goes here

// -->
</SCRIPT>

</HEAD>
<BODY>
HTML page text

<SCRIPT LANGUAGE="JavaScript">
<!–

//JavaScript goes here too

// -->
</SCRIPT>

HTML page text
</BODY>
</HTML>
```

Notice that the HTML comment tag is used to "comment out" or hide the text of the script. You use this feature in case the person browsing your page has an old browser that cannot execute JavaScript code. The comment tags prevent the specified code from displaying in the browser window. As more users upgrade their browsers, the comment tag will become less necessary for this purpose.

Notice the special addition in the ending comment tag line. The // characters at the beginning of the line are how JavaScript comments out text. JavaScript does not know how to handle the ending comment tag because it consists of two JavaScript operators; therefore you must hide that line from the JavaScript code. Failure to include those lines results in an error message when the page is loaded in some browsers, and may keep the script from executing properly.

INSTRUCTOR NOTE: Some older browsers will display JavaScript code even if the HTML comment tags are used. Also, stress the importance of placing the opening HTML comment tag after the opening <SCRIPT> tag and the closing HTML comment tag before the closing </SCRIPT> tag. Placing the HTML comment tags outside the <SCRIPT> </SCRIPT> tags will not work.

The HTML 4.0 TYPE attribute

The HTML 4.0 specification deprecated the LANGUAGE attribute for the <SCRIPT> tag. In its place, the TYPE attribute is recommended. This attribute is used as follows:

```
<SCRIPT TYPE="text/javascript">
```

The value for TYPE is in the form of a MIME type descriptor. Note that only browsers such as Netscape Navigator 6.0 and later, and Microsoft Internet Explorer 5.0 and later, support the TYPE attribute. The LANGUAGE attribute can be used as needed because it is still supported by all JavaScript-enabled browsers.

Denoting JavaScript versions

As noted earlier, several different versions of JavaScript exist. If you need to ensure that browsers know which version of JavaScript you are using, this information can be placed in the <SCRIPT> tag using the LANGUAGE attribute. The following example demonstrates this syntax:

```
<HTML>
<HEAD>
<TITLE>Page Title</TITLE>

<SCRIPT LANGUAGE="JavaScript1.5">
<!–

//JavaScript 1.5 code goes here

// -->
</SCRIPT>

</HEAD>
<BODY>
HTML page text
</BODY>
</HTML>
```

In this example, the developer forces the user agent to use the 1.5 version of JavaScript. If the user agent cannot execute JavaScript 1.5 code, the entire script block is ignored.

If the LANGUAGE attribute is omitted from the <SCRIPT> tag, Netscape Navigator and Microsoft Internet Explorer will typically assume the most current version of JavaScript.

External scripts

INSTRUCTOR NOTE: To use external scripts, the Web server must be able to map files with a .js extension to a MIME type of `application/x-javascript`. If external scripts are used, they should be thoroughly tested before deployment, with adjustments made to the Web server as needed.

You can also include script as an external file. This strategy is helpful if your code is more complex, if you plan on revising the code often, or if you plan on using the same code in multiple pages.

To create an external JavaScript file, use the .js file name extension. The .js file consists of native JavaScript code without the HTML <SCRIPT> tag. You can then include code similar to the following within the <HEAD> or <BODY> tags (note the use of the SRC attribute):

```
<SCRIPT LANGUAGE="JavaScript" SRC="JavaScriptCode.js">
<!–

/*
Avoid adding embedded JavaScript code. It will
   not be executed if the .js file is unavailable.
*/

// -->
</SCRIPT>
```

The browser will automatically read the code written in the .js file as if it were placed between the <SCRIPT> tags. As noted in the code, any additional embedded JavaScript code will not be executed if the .js file is not available in Netscape Navigator 4.x and later and in Microsoft Internet Explorer 4.x and later. If any additional code needs to be executed in the file, that code should reside in a different <SCRIPT> block. The JavaScript comment tags used in the previous examples will be discussed in detail in the next section.

WARNING! *Note that Netscape Navigator 2.x and Microsoft Internet Explorer 3.x do not support external JavaScript files. Use caution if your target audience uses these browsers. Also note that the Web server hosting the application must be aware of the .js extension to associate the proper file with the HTML page.*

The <NOSCRIPT> tag

You should be aware that your users may have older browsers that do not support JavaScript, or may disable execution of JavaScript code within their browsers. If so, you can use the HTML <NOSCRIPT> tag to render content for these users. An example of the <NOSCRIPT> tag syntax follows:

```
<SCRIPT LANGUAGE="JavaScript">
<!–

//JavaScript to insert information

//-->
</SCRIPT>
<NOSCRIPT>
 Click here for
<A HREF="http://www.yourCompany.com/info/info.htm">info.</A>
</NOSCRIPT>
```

Annotating Your Code with Comments

INSTRUCTOR NOTE: Adding comments to scripts as they are being built is an important element of the development process. However, though the JavaScript interpreter ignores all comments, the comment text is still part of the downloadable document. Comments should be minimized after the script is operational.

As your scripts become longer and more complex, they will become harder to read and debug. For this reason, successful programmers place comments in their code to help remind them of the purpose and function for each major section of code.

You know that HTML allows you to use comment tags to place comments in the document that have no effect on the document's appearance to the user. Similarly, you can also place comments in your JavaScript code.

JavaScript uses two types of comment indicators. One indicator delineates a comment on a single line of script (//). The other type of comment indicator is used for multiple-line comments (/*...*/).

Single-line comment indicator

You have already used the // to comment out the ending HTML comment tag. However, the single-line comment indicator is not limited to this use. You can use this method to add comments to a whole or partial line, as demonstrated in the following code excerpt:

```
<SCRIPT LANGUAGE="JavaScript">
<!--

// Variables defined here

var firstNum = 20
var secondNum = 0   // this value will change

//-->
</SCRIPT>
```

In this example, the portion of the coding from the // to the end of the line will be ignored by the JavaScript interpreter.

Multiple-line comment indicator

Eventually, you will need to use a comment that extends beyond a single line. To perform this task, enclose the area that will not execute with the /* and */ indicators. Note that the syntax for this comment is exact.

The following is an example of a multi-line comment:

```
<SCRIPT LANGUAGE="JavaScript">
<!--

/*
   The function addNumbers( ) is used to calculate the
    two numbers that are supplied to the function by the
    user.
*/

function addNumbers() {
  //code for function
}

//-->
</SCRIPT>
```

Tech Note

You can also use comments to prevent a section of code from executing if you need to troubleshoot your script. If you enclose the section of suspect script within comment indicators, JavaScript will ignore that section when your script executes.

The following lab will provide an opportunity to create a JavaScript-enabled HTML page. Any text editor can be used to create this page. Check with your instructor for special instructions concerning the editor to be used for this course.

Before beginning this lab, copy the Student_Files folder from the supplemental CD-ROM to your desktop.

Lab 1-1: Creating a JavaScript-enabled page

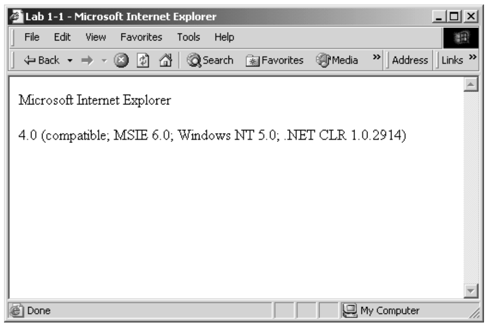
In this lab, you will create your first JavaScript page, which will introduce two JavaScript objects using a method of one and two properties of the other. The first object is the `document` object and will use its `write` method. The second object is the `navigator` object and will use its `appName` and `appVersion` properties.

1. **Editor:** Open the **lab1-1.htm** file from the Lesson 1 folder of the Student_Files directory. Enter the code indicated in bold:

```
<HTML>
<HEAD>
<TITLE>Lab 1-1</TITLE>
</HEAD>

<BODY>

<!-- Create <SCRIPT> block here -->

<SCRIPT LANGUAGE="JavaScript">
<!--

document.write(navigator.appName);
document.write("<P>");
document.write(navigator.appVersion);


//-->
</SCRIPT>

</BODY>
</HTML>
```

2. **Editor:** Save **lab1-1.htm**.

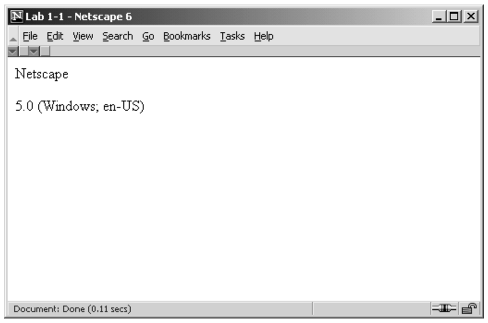
3. **Browser:** Browse to **lab1-1.htm**. If you are not sure how to access the file, ask your instructor for help. Your screen should resemble Figure 1-2, depending on the browser you are using for this course. Figure 1-2 shows lab1-1.htm in Microsoft Internet Explorer 6.0.


Lab 1-1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media Address Links
Done
My Computer

```
Microsoft Internet Explorer

4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.0.2914)
```

Figure 1-2: Lab1-1.htm in Internet Explorer 6.0

4. **Browser:** Figure 1-3 shows lab1-1.htm in Netscape Navigator 6.2.


Lab 1-1 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
Document: Done (0.11 secs)

```
Netscape

5.0 (Windows; en-US)
```

Figure 1-3: Lab1-1.htm in Netscape Navigator 6.2

As you can see, differences exist in the format that each browser uses for the output of the JavaScript statements. This example indicates the differences in implementing JavaScript from browser to browser.

In this lab you used a `document.write()` statement. The `document` object's `write()` method is used to output data to the HTML stream. You also used the `navigator` object's `appName` and `appVersion` properties. The `appName` property returns a string value indicating the name of the client browser. The `appVersion` property returns a string value indicating the version number of the client browser, as well as the client's platform.

Notice that in the `document.write()` statements, the text `navigator.appName` and `navigator.appVersion` were not inside quotation marks, whereas the HTML <P> tag was inside quotation marks. The two lines of code using the `navigator` object are evaluated text. In other words, the JavaScript interpreter dynamically supplies the appropriate text when the script is executed. Therefore, that text was not inside quotation marks. The literal <P> tag is static text. Its value is known before the script runs, so it is placed inside quotation marks.

Lesson Summary

Application project

The World Wide Web provides many valuable JavaScript resources. Use any good search engine and locate several JavaScript resources. Also, resources are listed in an appendix. Begin investigating sites that provide language documentation, tutorials and code examples. Become familiar with the various resources on the World Wide Web. Determine and document which sites provide the most information, which sites are the easiest to use, and which will benefit you the most as you learn the JavaScript language.

Skills review

Computer users' expectations and the demand for sophisticated graphical application interfaces have resulted in the advent of powerful scripting languages, such as JavaScript. JavaScript was first supported in Navigator version 2.0 and has since gained wide support in most popular browsers. Like other scripting languages, JavaScript is interpreted, not compiled to any particular machine or operating system. Scripting languages are subsets of larger, more complicated languages. JavaScript is an object-based language because it derives functionality from a collection of built-in objects. JavaScript is event-driven and platform-independent. In programming, objects encapsulate predesignated attributes and behavior. Developers refer to attributes and behaviors by three specific terms: properties, values and methods. Properties represent various attributes of an object. Values represent the specific qualities of properties. Methods are the actions that an object can be made to perform. Although the names are similar, Java and JavaScript are different languages. Java is a full-fledged object-oriented programming language developed by Sun Microsystems. JavaScript is an object-based scripting language that stands alone and does not require Java. Currently, six versions of JavaScript exist. JavaScript can be used for client-side and server-side solutions. JavaScript is usually placed within an HTML document using the `<SCRIPT>` tags. You can also include JavaScript as an external file. JavaScript uses two types of comment indicators, one for single line comments (//) and one for multiple-line comments (`/*...*/`).

Now that you have completed this lesson, you should be able to:

- ✓ Describe the origins of JavaScript.
- ✓ List the key JavaScript characteristics.
- ✓ Describe the differences between Java and JavaScript.
- ✓ Discern among JavaScript, JScript and VBScript.
- ✓ Differentiate between server-side and client-side JavaScript applications.
- ✓ Embed JavaScript into HTML.
- ✓ Use the JavaScript comment tags.

Lesson 1 Review

1. What advantage do scripting languages such as JavaScript offer over HTML?

   Scripting languages offer increased user interaction.

2. What advantage do scripting languages such as JavaScript offer over programming languages such as Java?

   Scripting languages offer extended functionality, but are easier and quicker to learn than full programming languages.

3. Name at least three key characteristics of JavaScript.

   JavaScript is a scripting language; JavaScript is object-based; JavaScript is event-driven; JavaScript is platform-independent; JavaScript enables quick development; JavaScript is easy to learn.

4. Where does client-side JavaScript code reside, and how is it activated?

   Client-side JavaScript code resides within an HTML document, and in some cases is embedded directly within certain HTML tags (which is called inline scripting). JavaScript is activated by an event, to which the script is instructed to respond with a designated action.

5. In programming, what is an object?

   An object is a programming function that models the characteristics of abstract or real "objects." An object encapsulates predesignated attributes and behaviors, and is often grouped with similar objects into classes.

6. In programming, developers refer to attributes and behaviors by what other three terms?

   Properties, values and methods.

7. How do JavaScript, VBScript and JScript differ?

   VBScript is similar in purpose to JavaScript. It was created by Microsoft as a subset of the Visual Basic programming language. VBScript relies more on customized functions and is supported only by the Microsoft Internet Explorer browser. JScript is the Microsoft version of Netscape's JavaScript. Minor differences exist in the implementations of JavaScript and JScript, with some differences causing compatibility problems.

8. What is the name of JavaScript's server-side technology?

   LiveWire.

9. What two types of comment indicators can you use to annotate your JavaScript code?

   Single-line comments (//) and multiple-line comments (/...*/).*

Lesson 1
Instructor Section

This section is a supplement containing additional tasks for students to complete in conjunction with the lesson. It also contains additional instructor notes. The instructor may use all, some or none of these additional tools, as appropriate to the specific learning environment. These elements are:

- **Additional Instructor Notes**
  Detailed instructor notes that expand the tips and suggestions presented in the instructor margin notes throughout the lesson.
- **Optional Labs**
  Computer-based labs to be completed during class or as homework.
- **Lesson Quiz**
  Multiple-choice test to assess student knowledge of lesson material.

Additional Instructor Notes

The following section contains detailed instructor notes that expand the tips and suggestions presented in the instructor margin notes throughout the lesson.

Instructor Note 1-1

Location: JavaScript and Common Programming Concepts

This instructor note provides another example of object, methods and properties.

Remember that objects are reusable units of code that have behaviors and attributes. An object's behaviors are generally referred to as *methods*, and its attributes are referred to as *properties*. An object's methods determine the actions the object can perform. Its properties can be used to set or determine the object's state. An object can also contain *collections* of properties, generally in key/value pairs.

Think of a car as an object. A car object might have the following properties:

```
ObjectCar.Make
ObjectCar.Model
ObjectCar.Year
ObjectCar.Color
ObjectCar.Price
```

These properties can set or check the car object's state. Some properties might describe the object's current state: `ObjectCar.Location` is an example of a car object's property that could refer to its current state.

The car object's speedometer, oil pressure gauge and temperature gauge can represent a collection of properties. Remember that key/value pairs are used to define collections. With the car object, the following key/value pairs can be used:

```
Key             Possible Value
Speedometer     65 mph
Oil Pressure    160 lbs
Temperature     190 degrees
```

The car object might contain the following methods:

```
ObjectCar.Drive()
ObjectCar.Accelerate()
ObjectCar.TurnLeft()
ObjectCar.TurnRight()
ObjectCar.Brake()
```

These methods indicate the actions the car object is made to perform. An object reveals itself through its public methods. These public methods form the object's interface. Using the car object, two public methods would be `drive` and `accelerate`, which are considered public methods because they can be accessed without going into the car object's inner workings. An object's inner workings represent its private methods. Using the car object, examples of private methods might be the interoperation of the pistons, rotor and camshaft. The car's driver need not be concerned with how these private methods operate, as long as the car drives and accelerates.

Objects used in programming operate similarly. The object's properties and methods only need to be accessed, used and reused when necessary, without concern for the object's inner workings.

Think of objects as small software packages that can be used in various situations and in different places in an application. Reusing objects is one of the primary advantages of object-oriented and object-based programming.

Instructor Note 1-2

Location: Java and JavaScript

This instructor note lists two URLs that will provide information concerning JavaScript and JScript versions.

For information concerning JavaScript versions:

http://developer.netscape.com/docs/manuals/js/core/jsguide15/preface.html#1001681

For information concerning JScript versions:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/js56jsoriversioninformation.asp

Optional Lab 1-1: Experimenting with JavaScript

This lab will provide an opportunity to experiment with the JavaScript-enabled file that you created in Lab 1-1.

1. **Editor:** Open the **lab1-1.htm** file. Save the file as **optionalLab1-1.htm**. Experiment with the `document.write()` statements by adding literal text to them. The `document.write()` statement can receive text or any HTML that you may want to use. At this point, there is no right or wrong action to take. Experiment with the document. It will help you become familiar with adding or deleting code inside a JavaScript block.
2. **Editor:** After each change to the optionalLab1-1.htm file, save the file.
3. **Browser:** Browse to **optionalLab1-1.htm**. View your changes. If an error occurs, return to the file in your editor and see whether you can determine why the error occurred.

This lab provided you with an opportunity to experiment with a `document.write()` statement. You added text and HTML to the statements and viewed the output using the browser of your choice.

Lesson 1 Quiz

1. What is a scripting language?
   a. <u>*Subset of a full programming language*</u>
   b. Meta-language used to create other languages
   c. Markup language used to format text and graphics
   d. Programming language used for dialog-based functions

2. The JavaScript language was developed by which of the following?

   a. Microsoft
   b. Sun Microsystems
   c. *Netscape Corporation*
   d. World Wide Web Consortium (W3C)

3. A key characteristic of JavaScript is that it is:

   a. platform-dependent.
   b. object-oriented.
   c. *event-driven.*
   d. difficult to learn.

4. Which of the following accurately describes a difference between JavaScript and Java?

   a. JavaScript is object-oriented whereas Java is object-based.
   b. *JavaScript is interpreted by the client whereas Java is precompiled on the server.*
   c. JavaScript uses static binding whereas Java uses dynamic binding.
   d. JavaScript requires strong typing whereas Java allows loose typing.

5. Which task is an example of a server-side JavaScript application?

   a. Working with frames
   b. Launching a new window from a Web page
   c. Validating fields of data in a user-submitted Web form
   d. *Parsing and disseminating user-submitted information*

6. Which task is an example of a client-side JavaScript application?

   a. *Sending a message to a Web page user*
   b. Saving client state in a multi-page process
   c. Enabling a server-side image map
   d. Connecting a Web page to a database

7. Which tag is used to embed JavaScript into an HTML document?

   a. The HTML `<LANGUAGE>` tag
   b. The JavaScript `<SCRIPT>` tag
   c. The HTML `<JAVASCRIPT>` tag
   d. *The HTML <SCRIPT> tag*

8. Which syntax indicates a multiple-line comment in JavaScript?

   a. */*…*/*
   b. `<!--…-->`
   c. `//`
   d. `//-->`

Lesson 2: Working with Variables and Data in JavaScript

Objectives

By the end of this lesson, you will be able to:

- Communicate with users through the `alert()`, `prompt()` and `confirm()` methods.
- Define variables.
- Define data types.
- Obtain user input and store it in variables.
- Report variable text to the client window.
- Discern between concatenation and addition.
- Use expressions.
- Use operators.
- Define inline scripting.
- Implement simple events such as `onLoad()` and `onUnload()`.
- Define keywords and reserved words.

Pre-Assessment Questions

1. Which of the following JavaScript code blocks correctly scripts an `alert()` method?

a.
```
<SCRIPT LANGUAGE="JavaScript">
<!--
   document.alert("This is a message.");
//-->
</SCRIPT>
```

b.
```
<SCRIPT LANGUAGE="JavaScript">
<!--
   alert("This is a message.");
-->
</SCRIPT>
```

c.
```
<SCRIPT LANGUAGE="JavaScript">
<!--
   window.alert(This is a message.);
//-->
</SCRIPT>
```

d.
```
<SCRIPT LANGUAGE="JavaScript">
<!--
   window.alert("This is a message.");
//-->
</SCRIPT>
```

2. Which of the following JavaScript code blocks correctly scripts a `prompt()` method?

a.
```
<SCRIPT LANGUAGE="JavaScript">
<!--
   prompt(Please input data., default data);
//-->
</SCRIPT>
```

b.
```
<SCRIPT LANGUAGE="JavaScript">
<!--
   prompt("Please input data.", "default data");
//-->
</SCRIPT>
```

c.
```
<SCRIPT LANGUAGE="JavaScript">
<!--
   prompt("Please input data." "Default data");
//-->
</SCRIPT>
```

d.
```
<SCRIPT LANGUAGE="JavaScript">
<!--
   prompt("Please input data.");
//-->
</SCRIPT>
```

3. What is the return value of an `alert()` method?

   There is no return value from an `alert()` method. The method is used to display a message to the user. When the user clicks the OK button on the alert dialog box, the box closes without returning a value.

Using JavaScript to Communicate with the User

Variables, data types, expressions and operators are the fundamental concepts in any programming language, including JavaScript. Any discussion of a Web scripting language should begin with the simplest concepts, such as storing data in variables and displaying the contents of these variables to the user.

In many Web applications, you occasionally see message windows alerting you to changing conditions, choices, warnings or other notices.

As a Web page developer, you can quickly and easily script messages that will appear or "pop up" for the user, either when the page loads or upon some other event, such as a mouse click. For now, you will use the primary event—the loading of the page into the window—to launch scripts. In this lesson, you will examine four ways of communicating with the user: a text message that displays in a pop-up window, a request for information in a pop-up window, a confirmation request in a pop-up window, and a text message that displays in the browser window. Each of these methods of communication will show how to manipulate various JavaScript objects, as well as how to name and use variables.

As you learn about various methods such as `alert()`, `prompt()` and `confirm()`, remember that these JavaScript implementations all rely on larger concepts. For example, `alert()`, `prompt()` and `confirm()` are methods of the `window` object. You will also learn how to use the `alert()` method with variables, which are named spaces of memory.

Giving the user a message: The *alert()* method

INSTRUCTOR NOTE: Refer to **Instructor Note 2-1** for more information concerning the `alert()` method.

The `alert()` method is simple JavaScript that allows you to communicate with the user.

statement
A single line of code to be executed in a script or program.

To call the `alert()` method, you need only place a simple line of code called a **statement** in a script block somewhere in your document. If you have no other code in your script, the message will pop up every time the page is loaded. If other code is in the script, the message window will pop up when it is called in the code sequence. This process occurs because the `alert()` method always implies the use of the `window` object.

INSTRUCTOR NOTE: This section demonstrates how to create an alert dialog box. Students will apply this information in the next lab. They will not copy code. This section is the students' reference point if they need to see how to perform this task.

The syntax for using the `alert()` method is as follows:

```
alert("message");
```

In this example, the text `message` is given as the argument to the `alert()` method. If the code in this example were executed, a dialog box with the word `message` would appear on the user's screen.

WARNING!

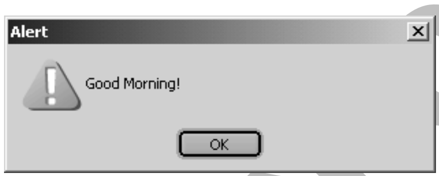
Note that the alert dialog box is modal in nature. This means that the user must acknowledge the dialog box by clicking the OK button or pressing ENTER before being allowed to navigate further with the present browser window. For this reason, use JavaScript dialog boxes with caution. Overuse of alerts could deter your users from accessing your Web application.

In the following lab, you will create a script using the `alert()` method. This lab demonstrates a script with a single statement.

Lab 2-1: Using the JavaScript *alert()* method

In this lab, you will use the JavaScript `alert()` method to give a message to the user.

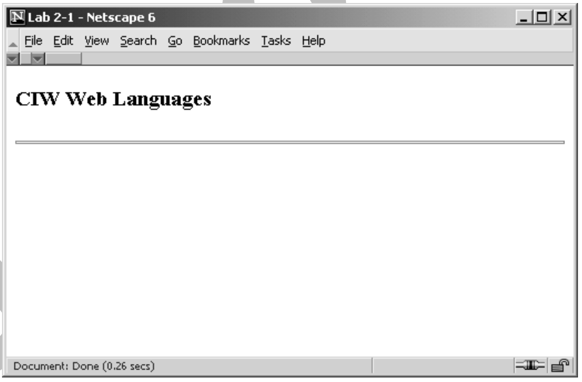
1. **Editor:** Open the **lab2-1.htm** file from the Lesson 2 folder of the Student_Files directory.
2. **Editor:** Locate the `<SCRIPT LANGUAGE="JavaScript"> </SCRIPT>` block in the `<HEAD>` section of the document. Within the block, add an **`alert()`** with the message `"Good Morning!"` as the argument.
3. **Editor:** Save **lab2-1.htm**.
4. **Browser:** Open **lab2-1.htm**. You should see a dialog box that resembles Figure 2-1. If you do not, verify that the source code you entered is correct.



Good Morning!

Figure 2-1: Alert message

5. **Browser:** After you click **OK**, your screen should resemble Figure 2-2.



CIW Web Languages

Document: Done (0.26 secs)

Figure 2-2: Lab2-1.htm displayed following JavaScript statement

Alert messages can be used to display any information you want, such as copyrights, current date and time, notices of site changes, or a quote of the day. These messages can be very useful when you need to debug your own script. Throughout this course, you will learn to make a client browser perform increasingly sophisticated tasks.

Using semicolons in JavaScript

Notice that in the example preceding the lab, the `alert()` statement was followed by a semicolon. The semicolon is often used in programming to separate statements from each other. Unless you are combining statements, the semicolon is not necessary in JavaScript. However, many programmers consider it good practice to use the semicolon after every statement. This practice is especially useful—and sometimes necessary—if you have multiple statements in one line of the source code.

Getting data from the user: The *prompt()* method

INSTRUCTOR NOTE: The information in **Instructor Note 2-1** concerning the title bar of an alert dialog box also applies to the title bar of a prompt dialog box.

You may want to request information from the user and capture his or her reply. This task requires another method of the `window` object.

The `prompt()` method requests user input through a text field within a dialog box. Whatever the user types into the text box is returned as the result of the `prompt()` method in the form of a manipulatable text string, which you can use as an argument to another method, or in any other way you see fit. The basic syntax for using `prompt()` is as follows:

```
prompt("Message to user", "default response text");
```

In this example, the text `Message to user` will appear above the text entry area of the `prompt()` dialog box. The text `default response text` will appear in the text-entry area. If you do not want default text in the text-entry area, you must follow the initial message with a comma and an empty set of quotation marks. This syntax is required for the `prompt()` method.

Concatenation

concatenation
Synthesis of code to simplify it and reduce duplication; linking two or more units of information, such as strings or files, to form one unit.

Concatenation is used frequently in JavaScript to combine text strings, especially in conjunction with the `prompt()` and `alert()` methods. Because the `prompt()` method always returns a string, concatenation allows you to combine strings for further manipulation. For example, you may want to personalize the alert message in the previous lab. To do this, you can concatenate the "Good Morning!" message with the result of a prompt screen in which the user enters his or her name. Following is an example of this type of syntax:

```
alert("Begin message " + prompt("Message to user", ➘
      "default text") + " End message.");
```

In this example, the processing of the `prompt()` method will take priority over the processing of the `alert()` method. A method that is defined inside another method will always execute before the outer method. The syntax of the `prompt()` method will be discussed further following the lab, in which you will use concatenation.

INSTRUCTOR NOTE: See **Instructor Note 2-2** for important information concerning the example in this section.

In the following lab, you will prompt the user for his or her name. You will then tie that name, using the + operator, to the phrase "Good morning, ". When the user enters a name, a message will be displayed saying "Good morning, *name*." including the name entered by the user.

You will learn more about other JavaScript operators later in this lesson.

Lab 2-2: Using the JavaScript *prompt()* method

In this lab, you will use the JavaScript `prompt()` method with concatenation to request and capture user input.

1. **Editor:** Open **lab2-2.htm** from the Lesson 2 folder of the Student_Files directory.

2. **Editor:** Locate the `alert()` method that has been defined for you. Modify the source code by adding a **`prompt()`** method that asks for the user's name. Concatenate the user's input with the existing text and add a closing period after the user input.

3. **Editor:** Save **lab2-2.htm**.

4. **Browser:** Open **lab2-2.htm**. When the page loads, you should see a prompt dialog box that resembles Figure 2-3. If not, verify that the source code you entered is correct.

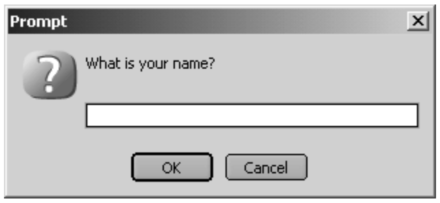
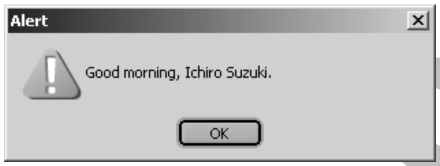


Figure 2-3: User prompt dialog box

5. **Browser:** Enter your name in the text field, and then click **OK**. Your screen should display a message that resembles the one shown in Figure 2-4.



Good morning, Ichiro Suzuki.

Figure 2-4: Alert message box

6. **Browser:** When you click **OK**, your screen should resemble Figure 2-2 on a previous page

In this lab, the `prompt()` method is processed first and the user's input is then concatenated into the expression. In other words, the `prompt()` method will take precedence over the `alert()` method in a JavaScript statement. In fact, JavaScript statements always execute from the inside out.

INSTRUCTOR NOTE: The JavaScript parentheses operators take precedence over all other operators. Thus, when any JavaScript statement, expression or method is enclosed within parentheses and embedded inside another set of parentheses, the inner code will always execute before the outer code.

In the preceding lab, you created some code that resembles the following:

```
alert("Begin message " + prompt("Message to user", ➘
      "default text") + " End message.");
```

Because the `prompt()` method is embedded inside the `alert()` method, the JavaScript interpreter will execute the `prompt()` method first, then execute the `alert()` method when the `prompt()` method has finished executing.

As previously mentioned, the syntax requirements of the `prompt()` method dictate that the argument is supplied in two parts. If you do not want default text in the text-entry area, you still must follow the initial message with a comma and an empty set of quotation marks, as shown in the following example:

```
prompt("What is your name?","");
```

If time allows, try entering different text within both sets of quotation marks to see the effect on the display.

INSTRUCTOR NOTE: The information in **Instructor Note 2-1**, concerning the text in the title bar of an alert dialog box, also applies to the title bar of a confirm dialog box.

Requesting confirmation: The *confirm()* method

Often in programming, you will want to ask the user a simple yes-or-no question. You can do this with a message box that provides both OK and Cancel buttons, using the `confirm()` method. It is similar to the `alert()` method with one significant exception: The `confirm()` method returns a value of either true or false, whereas the `alert()` method has no return value. This return value can be captured and used in your scripts. The concept of return values will be explored later in this course.

Like the `prompt()` method, the `confirm()` method takes processing precedence over the `alert()` method if both are used in the same expression.

The syntax for the `confirm()` method is as follows:

```
confirm("Question to the user?");
```

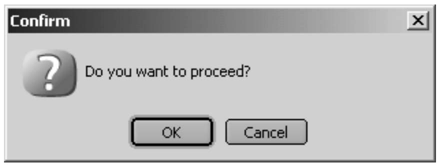
In the following lab, you will capture the return value of a `confirm()` dialog box.

INSTRUCTOR NOTE: See **Instructor Note 2-3** for important information concerning the example in this section.

Lab 2-3: Using the JavaScript *confirm()* method

In this lab, you will use the `confirm()` method to elicit a true or false return value from the user.

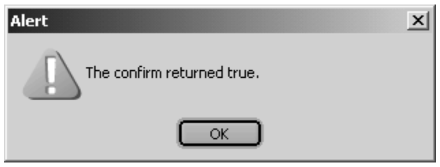
1. **Editor:** Open **lab2-3.htm** from the Lesson 2 folder of the Student_Files directory.
2. **Editor:** Locate the `alert()` method that has been defined for you. Modify the code to use the return value of a `confirm()` dialog box as the text for an `alert()` dialog box. In other words, concatenate the result of a `confirm()` method into an `alert()` method just as you previously concatenated the result of a `prompt()` method into an `alert()`. The argument for the `confirm()` method should read **`Do you want to` proceed?**
3. **Editor:** Save **lab2-3.htm**.
4. **Browser:** Open **lab2-3.htm**. When the page loads, you should see a confirm dialog box that resembles Figure 2-5. If not, verify that the source code you entered is correct.


Confirm

Do you want to proceed?

Figure 2-5: Confirm dialog box

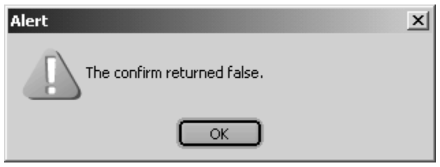
5. **Browser:** Click **OK**. Your screen should resemble Figure 2-6.


Alert

The confirm returned true.

Figure 2-6: Result of `confirm()` method after clicking OK

6. **Browser:** Reload **lab2-3.htm**. Click **Cancel**. Your screen should resemble Figure 2-7.



The confirm returned false.

Figure 2-7: Result of `confirm()` method after clicking Cancel

event handler
A JavaScript mechanism for intercepting an event and associating executable code with that event.

The return value of the `confirm()` method can be used in many different situations. For example, a `confirm()` can be used to validate form actions. If you want to help a user make an accurate choice, you can place a form **event handler** called `onSubmit` in the opening `<FORM>` tag. For example:

```
<FORM METHOD="post" ACTION="appropriate url"
onSubmit="return confirm('Submit now?');">
```

INSTRUCTOR NOTE:
The example in this section uses a set of single quotation marks inside a set of double quotation marks. Be prepared to answer questions concerning this syntax. This syntax is discussed later in this lesson as well as in other sections of later lessons.

When a user clicks the Submit button on such a form, the `onSubmit` event handler is invoked in response to that event. At that point, the `confirm()` method will present the user with a dialog box asking if the form should be submitted. If the user selects the OK option, a true value will be returned and the form will be submitted to the specified action. If the user chooses the Cancel button, a false value will be returned and the script will prohibit submission of the form.

Event handlers such as `onSubmit` are discussed in this course. The previous example also introduced the `return` keyword. As its name implies, this keyword is used in JavaScript to return values in various situations.

Writing HTML dynamically: The *document.write()* method

INSTRUCTOR NOTE:
This section demonstrates how to embed a `prompt()` method into a `document.write()` statement. Students will apply this information in the next lab.

Thus far in this lesson, you have created simple windows to alert and prompt your users. The previous lesson briefly introduced another way to communicate with the user. This JavaScript mechanism is to write text into the document as the page loads in the browser. To perform this task, you use the `document.write()` method, which belongs to the `document` object. The `document.write()` method allows you to create text that is dynamically written to the window as the script is executed. You can write out plain text or you can mix HTML tags with the text being written.

The basic syntax of the `document.write()` method is as follows:

```
<HTML>
<HEAD>
<TITLE>Example</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!--

    document.write("How wonderful to meet you, " + ➘
    prompt("Hello, my name is James. What's yours?", "")➘
     + ".");

//-->
</SCRIPT>
</BODY>
</HTML>
```

This syntax is different from `alert()` and `prompt()` in that it includes the specification of a new object, called `document`. The first three methods you used—`alert()`, `prompt()`

and confirm()—were methods of the window object in the JavaScript language. Those methods could also have been written as follows:

```
window.alert();

window.prompt();

window.confirm();
```

dot notation
A reference used to access a property or method of an object.

Because the window object is the default object, specifying window before each method is not necessary. But now you are using the write() method of the document object, which is not the default object and therefore needs to be referred to explicitly. You make this reference using **dot notation**. Typical syntax in dot notation is as follows:

```
objectName.methodName();

objectName.propertyName;
```

You will learn more about dot notation throughout this course. For now, you need to know that by scripting document.write("Hello") you are writing the contents of "Hello" directly to the page as it loads.

The next lab demonstrates the use of document.write(). It is similar to Lab 2-2, but uses document.write() instead of alert() to return text to the browser screen. A portion of the text will be collected from the user with the prompt() method.

Lab 2-4: Using the JavaScript *document.write()* method

INSTRUCTOR NOTE: Students may ask what makes a script execute, or when does a script execute. The JavaScript script engine interprets all JavaScript code. Also, there are four times when a script will execute: as a document loads in the browser, directly after a document loads, as a response to a user's action, or when invoked by other script statements.

In this lab, you will use the document.write() method to customize a page for the user.

1. **Editor:** Open **lab2-4.htm** from the Lesson 2 folder of the Student_Files directory.

2. **Editor:** Locate the prompt() method that has been defined for you. Modify the source code to use a **document.write()** statement. Concatenate the results of the prompt() method into the **document.write()** expression. Designate the output of the document.write() as an <H4> level greeting that displays the following:

   Welcome, *user's name*.

   The text *user's name* will be the return value from the prompt() method. Be sure to end with a period and close the <H4> tag after inserting the user's name.

3. **Editor:** Save **lab2-4.htm**.

4. **Browser:** Open **lab2-4.htm**. You should see a dialog box that resembles Figure 2-8.

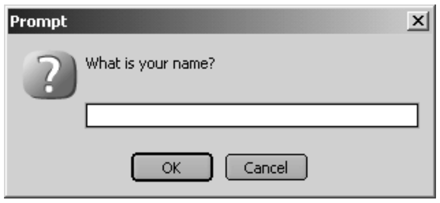
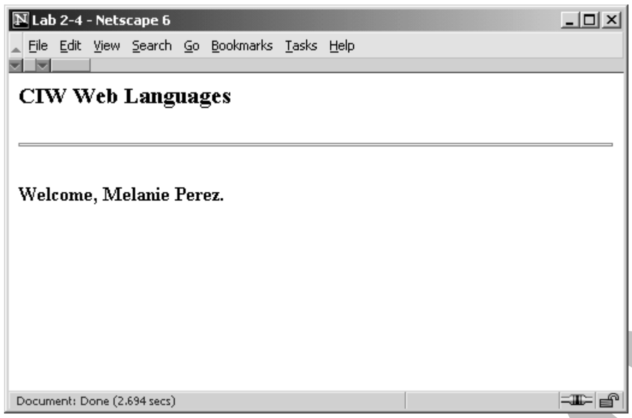


Figure 2-8: User prompt

5. **Browser:** Type your name in the dialog box, and then click **OK**. Your screen should resemble Figure 2-9.


Lab 2-4 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
Document: Done (2.694 secs)

CIW Web Languages

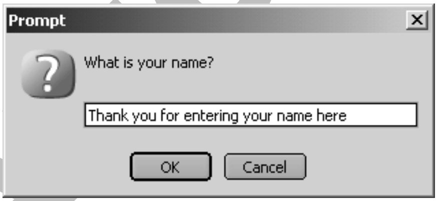
Welcome, Melanie Perez.

Figure 2-9: Lab2-4.htm with customized welcome message

6. Edit the initial prompt to contain a message within the text entry field. Place the text string you want inside of the now-empty quotation marks. For example, you could insert the following text shown in bold:

```
prompt("Hello! Please type your name in the box.","Thank you ➘
for entering your name here");
```

This addition would alter the initial alert box as shown in Figure 2-10.



What is your name?

Thank you for entering your name here

Figure 2-10: Customizing initial prompt

Note that in this lab, you were able to include an HTML heading tag as part of the text that was written to the screen. HTML can be freely interspersed with text when both are using the `document.write()` method. Note also that the `prompt()` method takes processing precedence over the `document.write()` method when both are used in the same expression.

Using Data More Than Once: Variables

In the previous labs, the user's name was provided at the moment you needed it in the script sequence. When script manipulated the `window` object through certain methods (`alert()`, `prompt()` and so forth), the user was there to enter requested information. The way these labs are structured, you would have to repeat that script sequence if you wanted to see or use the name again. Instead, you can store the user's name and use it as needed. Once you obtain the user's name and store it for reuse, you could use the data in any way that you see fit. To store a name for reuse, you need to assign the name as the value for a variable.

What is a variable?

variable
A named space of memory that holds a value. The value can change depending on a condition, information passed to the program, or by reassignment of a new value.

A **variable** is a named space of memory. It is a container that holds a value, which you can then access repeatedly in your script. Like many other programming languages, JavaScript allows you to declare variables and use their stored values. The name "variable" is appropriate because the values you store will vary from one variable to the next. You should also note that a variable's value may change over the course of a script sequence.

INSTRUCTOR NOTE: JavaScript will attempt to treat variables as the data type that makes the most sense for each situation. For instance, if an attempt is made to add the integer 6 to the string "6" the result will be the string "66" instead of the integer 12. In fact, JavaScript will concatenate instead of add in that situation.

Variable data types

Variables can store five data types in JavaScript: number, string, Boolean, null and Object. Other languages use many more data types. Table 2-1 lists examples of the values these data types represent.

Table 2-1: JavaScript data types

Data Type	Examples
number	Any numeric value (e.g., 3, 5.3, or 45e8)
string	Any string of alphanumeric characters (e.g., "Hello, World!", "555-1212" or "KA12V2B334")
Boolean	True or false values only
null	A special keyword for the null value
Object	A reference to a JavaScript object

Literals

Literals are the actual data values you provide in JavaScript. These are fixed values that you assign to variables in your script. Table 2-2 lists three examples of literals used in JavaScript.

Table 2-2: JavaScript literal types

Literal Type	Example
Integer literal	56
Floating-point literal	23.45
String literal	"Howdy!"

When assigning literal values to variable names, you place string values inside quotation marks. Numeric values are not placed inside quotation marks.

Naming variables

Variables should have meaningful and descriptive names. Consider what each variable represents within the context of your script, then choose an appropriate name. Even though JavaScript can be lenient when it comes to naming variables, you still must follow certain rules. Following are basic rules for naming variables in JavaScript:

- The first character of the variable must be a letter or the underscore (_) character. No other initial characters are allowed.
- Subsequent characters can be letters, numbers and/or underscore characters.

The following are examples of valid variable names:

- address1
- a4xb5
- lastName

The following examples would not be allowed as variable names because they use forbidden opening characters:

- 1stName
- ?subname
- @location

Tech Note *The JavaScript 1.5 specification allows variable names to begin with the dollar sign ($). Remember that the user must be utilizing a user agent that supports JavaScript 1.5, or the code block must be specified with a `LANGUAGE="JavaScript1.5"` attribute in the <SCRIPT> tag in order to use this new syntax.*

INSTRUCTOR NOTE: Many programmers prefer to place an underscore character (_) between the words of a multi-word variable name. This placement is acceptable in JavaScript.

The common naming convention in JavaScript is to use two words with no space between them, and capitalize the second word but not the first. For example, to store first names, you might use a variable called `firstName`.

WARNING! *JavaScript is case-sensitive, so the variable name "Result" is not the same as the variable name "RESULT." Whether you are declaring variables, defining functions, or otherwise using JavaScript, remember to be consistent about letter case. Many of your initial errors will come from improper case usage. Note, for example, that the `alert()` and `prompt()` methods of the `window` object are lowercase. You will receive an error if you type them otherwise.*

Declaring variables

INSTRUCTOR NOTE: This section demonstrates to students how to create variables and assign values to variables.

Although you need not explicitly declare variables in JavaScript, it is considered good programming practice to declare variables before they are used. Declaring a variable simply means telling JavaScript that a specific named variable exists.

In JavaScript, you can declare a variable using the `var` keyword:

```
var correctAnswer;
```

INSTRUCTOR NOTE: When a variable is declared and given a value in the same statement, the process is formally known as initializing a variable.

Here you have defined a variable named "`correctAnswer`", which does not hold any value. You can also assign an initial value to the variable when you declare it. For example:

```
var correctAnswer1 = 20;
var correctAnswer2 = "Yes";
```

Note the use of the equal sign (=) to assign the value to the variable. This character is the JavaScript assignment operator, and will be discussed later in this lesson along with other JavaScript operators.

INSTRUCTOR NOTE: The `var` keyword is only used for declaration or initialization of JavaScript variables. And the keyword is used only once per variable per document.

You can create and assign values to multiple variables with a single `var` keyword, as shown in the following example:

```
var correctAnswer1 = 20, correctAnswer2 = "Yes";
```

Note the use of a comma to separate the variable names in this example.

Note that variables can also receive values in other ways. For example, a `prompt()` or a `confirm()` method can be placed on the right side of a variable assignment expression. The return value of the method would be assigned as the value for the variable. A variable can also receive its value from another variable. Consider the following example.

```
var userName1 = "Juan";
var userName2 = userName1;
```

In this example, a variable is declared and initialized with a value. The second variable is assigned the value held in the first variable. This is perfectly legal syntax in JavaScript. Note that even if the value held in `userName1` changes later in the script, `userName2` would maintain the value originally assigned. To change the value of the `userName2` variable later in the script, a direct assignment statement would have to be executed.

As mentioned, JavaScript is case-sensitive, so when you declare a variable, use the name consistently or your script will not work properly.

Concatenating variables

INSTRUCTOR NOTE: This section demonstrates to students how to concatenate a variable into a `document.write()` statement.

When concatenating variables with literal strings, remember to place the variable name outside of any quotation marks. As you learned in another lesson, the JavaScript interpreter will dynamically supply the appropriate value when the script is executed. The following example demonstrates this concept:

```
var userName = "Julio";

document.write("We thank " + userName + " for visiting our site.");
```

Note that literal spaces are placed before and after the variable name in the preceding example. This ensures that the text will display properly.

Working with variables

In the previous labs, you used the `alert()` and `prompt()` methods to obtain the user's name and display it in the form of a greeting. Now, use your knowledge of variables to collect information from these methods and use it repeatedly.

In the following lab, you will capture the user's name only once but use it twice to conduct a short dialog.

Lab 2-5: Storing user data in a JavaScript variable

In this lab, you will store the user's name in a variable so you can use it as needed.

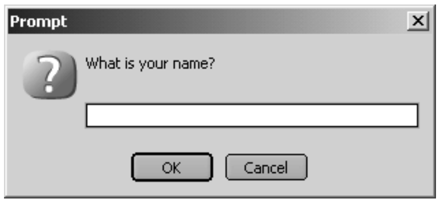
1. **Editor:** Open **lab2-5.htm** from the Lesson 2 folder of the Student_Files directory.

2. **Editor:** Locate the `prompt()` method that has been defined for you. Modify the source code to add a variable named **`userName`**. Assign the result of the `prompt()` method as the value for `userName`.

3. **Editor:** Concatenate the `userName` variable into the `alert()` and `document.write()` expressions that have been provided for you. Make sure to concatenate closing periods in both the `alert()` and `document.write()` statements.

4. **Editor:** Save **lab2-5.htm**.

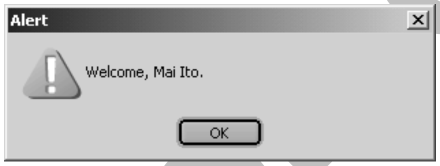
5. **Browser:** Open **lab2-5.htm**. Compare the prompt dialog box displayed in your browser with Figure 2-11. They should be similar.



What is your name?

Figure 2-11: User prompt dialog box

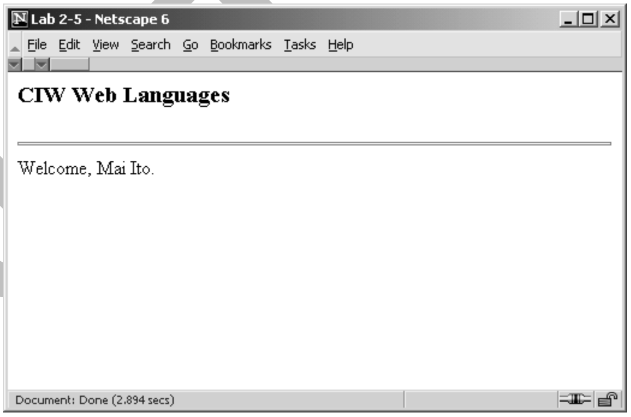
6. **Browser:** Enter your name in the text field and click **OK**. Your screen should display a message similar to Figure 2-12.



Welcome, Mai Ito.

Figure 2-12: Alert message box

7. **Browser:** Click **OK** again, and your screen should resemble Figure 2-13.



CIW Web Languages

Welcome, Mai Ito.

Document: Done (2.894 secs)

Figure 2-13: Lab2-5.htm with welcome message recalling user's name

Note: If you try to edit an HTML file with an alert message present in the browser, you may not be able to save your file until you acknowledge the message by clicking the OK button. If you do not acknowledge the alert, you might receive a "file sharing" violation from your operating system.

In this lab, you declared a variable called `userName`. In that variable, you stored the result of the `prompt()` command. The = character was used as the assignment operator.

INSTRUCTOR NOTE: Some common mistakes made when concatenating strings are: omitting one of the quotation marks around literal strings, forgetting to place blank spaces in literal strings for word spaces, and omitting punctuation after variable values.

You then concatenated the phrase "Welcome, " with the `userName` variable using the + operator, and displayed the result in an alert box. After the user acknowledged the first alert, you generated a `document.write()` statement, with text before and after the variable `userName`.

You might be wondering at this point if you can add numbers using the + operator because + is the operator for concatenation in text. Before you learn more about this, however, you must learn about expressions and operators.

JavaScript Expressions

An expression is a part of a statement that is evaluated as a value. An exception is an assignment expression, which assigns a value to a variable. An expression can use any combination of variables, literals, operators and other expressions. JavaScript uses the following four types of expressions.

- **Assignment:** assigns a value to a variable.

  Example: `myNumber = 25;`

  The value 25 has been assigned to the variable `myNumber`.

  Operator is = (assignment).

- **Arithmetic:** evaluates to a number.

  Example: `25 + 75;`

  This expression evaluates to the sum of 25 and 75.

  Operator is + (addition).

- **String:** evaluates to a string.

  Example: `"Hello, " + "Sandy";`

  This evaluates to a new string that says "Hello, Sandy".

  Operator is + (concatenation).

- **Logical:** evaluates to true or false.

  Example: `25 < 75;`

  Because 25 is less than 75, this expression evaluates to the Boolean value of true.

  Operator is < (less than comparison).

  Example: `25 >= 75;`

  Because 25 is not greater than or equal to 75, this expression evaluates to the Boolean value of false.

  Operator is >= (greater than or equal comparison).

Each of these expressions relies on an operator. You will now take a closer look at operators in JavaScript.

Operators

operand
Data that is to be operated upon or manipulated in some manner.

INSTRUCTOR NOTE:
Operators are the building blocks of expressions. An expression can contain just one operand and one operator. The operator in this case is referred to as a unary operator. Expressions can also contain two operands with an operator between them. The operator in this case is referred to as a binary operator. Note that JavaScript contains many more operators than are listed in this table. However, most operations can be carried out with the operators defined in Table 2-3.

In JavaScript, operators are used in expressions to store or return a value, generally manipulating **operands** in the process. An operand represents a piece of data that is manipulated in some manner. Operators come in many varieties, including assignment, arithmetic, unary and logical. Table 2-3 lists the most commonly used JavaScript operators and their uses. You will use many of these operators throughout the course.

Table 2-3: JavaScript operators

Operator	Type	Description	Examples
=	Assignment	Assigns the value of the right operand to the left operand	`x = 25;` `userID = "Juan";` x now equals 25. `userID` now equals Juan.
+=	Assignment	Adds together the operands and assigns the result to the left operand	`x = 25;` `x += 1;` x now equals 26.
-=	Assignment	Subtracts the right operand from the left operand and assigns the result to the left operand	`x = 25;` `x -= 1;` x now equals 24.
*=	Assignment	Multiplies together the operands and assigns the result to the left operand	`x = 25;` `x *= 2;` x now equals 50.
/=	Assignment	Divides the left operand by the right operand and assigns the result to the left operand.	`x = 25;` `x /= 2;` x now equals 12.5.
%=	Assignment	Divides the left operand by the right operand and assigns the remainder to the left operand	`x = 25;` `x %= 2;` x now equals 1.
+	Arithmetic	Adds the operands together	`x = 25;` `y = 10;` `z = x + y;` z now equals 35.
-	Arithmetic	Subtracts the right operand from the left operand	`x = 25;` `y = 10;` `z = x - y;` z now equals 15.
*	Arithmetic	Multiplies the operands together	`x = 25;` `y = 10;` `z = x * y;` z now equals 250.
/	Arithmetic	Divides the left operand by the right operand	`x = 25;` `y = 10;` `z = x / y;` z now equals 2.5.
%	Arithmetic	Divides the left operand by the right operand and calculates the remainder	`x = 25;` `y = 10;` `z = x % y;` z now equals 5.

Table 2-3: JavaScript operators (cont'd)

Operator	Type	Description	Examples

&&	Logical	The logical "and"; evaluates to true when both operands are true	`x = 25;`
			`y = 10;`
			`z = 35;`
			
			`(x > y) &&`
			`(y < z);`
			This statement returns `true`.
			`(x < y) &&`
			`(y < z);`
			This statement returns `false`.

||	Logical	The logical "or"; evaluates to true when either operand is true	`x = 25;`
			`y = 10;`
			`z = 35;`
			
			`(x > y) ||`
			`(y > z);`
			This statement returns `true`.
			`(x < y) ||`
			`(y > z);`
			This statement returns `false`.

!	Logical	The logical "not"; evaluates to true if the operand is false and to false if the operand is true	`x = 25;`
			`y = 10;`
			
			`!(x > y);`
			This statement returns `false`.
			`!(x < y);`
			This statement returns `true`.

==	Comparison	Evaluates to true if the operands are equal	`x = 25;`
			`y = 10;`
			
			`x == y;`
			This statement returns `false`.

!=	Comparison	Evaluates to true if the operands are not equal	`x = 25;`
			`y = 10;`
			
			`x != y;`
			This statement returns `true`.

>	Comparison	Evaluates to true if the left operand is greater than the right operand	`x = 25;`
			`y = 10;`
			
			`x > y;`
			This statement returns `true`.

<	Comparison	Evaluates to true if the left operand is less than the right operand	`x = 25;`
			`y = 10;`
			
			`x < y;`
			This statement returns `false`.

Table 2-3: JavaScript operators (cont'd)

Operator	Type	Description	Examples

`>=`	Comparison	Evaluates to true if the left operand is greater than or equal to the right operand	`x = 25;`
			`y = 10;`
			
			`x >= y;`
			This statement returns `true`.

`<=`	Comparison	Evaluates to true if the left operand is less than or equal to the right operand	`x = 25;`
			`y = 10;`
			
			`x <= y;`
			This statement returns `false`.

`+`	String	Combines the operands into a single string	`name1 = "Mary ";`
			`name2 = "Sue";`
			`name3 = name1 + name2`
			`name3` now contains
			`"Mary Sue."`

`(condition) ? val1 : val2`	Conditional	Evaluates to one of two different values based on a condition	`x = 25;`
			`y = 10;`
			
			`(x < y) ? true : false;`
			This statement returns `false`.

`++`	Unary	Increases the value of the supplied operand by one	`x = 25;`
			`x++;`
			`x` equals 26 the next time it is accessed.

`-`	Unary	Negates the value of the operand	`x = 25;`
			`x = -x;`
			`x` now equals -25.

`--`	Unary	Decreases the value of the supplied operand by one	`x = 25;`
			`x--;`
			`x` equals 24 the next time it is accessed.

Expressions rely on various operators. For example, + is the operator for addition as well as concatenation. The assignment operator is =, but it is never used as a comparison operator. For example, consider the following statement:

```
userName = "Nancy";
```

INSTRUCTOR NOTE: Note that the value on either side of an operator is referred to as an operand, which includes literal values that are assigned to variables.

Used in a script, this statement would assign the value of "Nancy" to the variable `userName`. If you wanted to compare `userName` to "`Nancy`", you would use the following code:

```
userName == "Nancy";
```

This expression compares whether the contents of `userName` are the same as the text string "`Nancy`" and would subsequently return a Boolean value of `true` or `false`, depending on the result of that comparison. If the value of the variable `userName` were "`Nancy`", then the value of `true` would be returned. If not, the value of `false` would be returned.

In this expression, both `userName` and `"Nancy"` are operands—in other words, data values that the script manipulates and processes.

Note that the increment (++) and decrement (--) operators can be placed either before or after an operand. The placement of the operator produces a subtle difference in its performance. If the increment or decrement operator is placed before the operand, the operation occurs immediately. If the increment or decrement operator is placed after the operand, the change in the operand's value is not apparent until the next time the operand is accessed in the program. Consider the following example:

```
var i = 1;
document.write("First access: " + i++);
document.write("<BR>Second access: " + i);
```

The output of the preceding piece of code would be as follows:

```
First access: 1
Second access: 2
```

As the preceding output demonstrates, the next time the variable `i` is accessed in the program, it contains the incremented value. Consider the following example:

```
var i = 1;
document.write(++i);
```

The output of this piece of code would be `2` because the increment operator is placed before the operand, causing the operation to occur immediately.

As you study JavaScript on your own, experiment with the various JavaScript operators. As you use them in your scripts, your familiarity with their varied functions will grow.

The following lab will demonstrate the use of the addition operator (+).

Lab 2-6: Assigning and adding variables in JavaScript

In this lab, you will assign variables and use the addition operator to add them together.

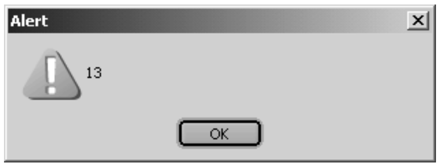
1. **Editor:** Open the **lab2-6.htm** file from the Lesson 2 folder of the Student_Files directory.

2. **Editor:** Add source code inside the `<SCRIPT LANGUAGE="JavaScript"> </SCRIPT>` block. Create two variables named **x** and **y**. Assign the numerical value of **4** to the first variable and **9** to the second variable.

3. **Editor:** Create a third variable named **z**. Assign to **z** the result of adding together **x** and **y**. Display the variable **z** in an `alert()` dialog box.

4. **Editor:** Save **lab2-6.htm**.

5. **Browser:** Open **lab2-6.htm**. Your screen should immediately display an alert box, as shown in Figure 2-14.



```
13
```

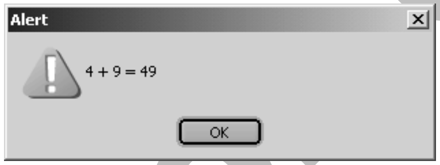
Figure 2-14: Result of JavaScript addition: z=13

6. **Editor:** Immediately after the `alert(z)` line, add the following source code:

```
alert("4 + 9 = " + x + y);
```

7. **Editor:** Save **lab2-6.htm**.

8. **Browser:** Reload **lab2-6.htm**. Your screen should display the alert box showing 13, as seen in Figure 2-14. When you click **OK**, you should then see another alert box as shown in Figure 2-15.



```
4 + 9 = 49
```

Figure 2-15: Concatenation, not sum

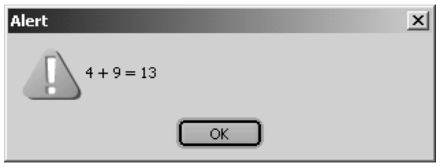
This alert shows that the + symbol was interpreted as an operator for concatenation. Because the values were concatenated, the script returned a result of 49. Note that this behavior is caused by the fact that the `alert()` method normally takes a string value as its argument. So the JavaScript interpreter treated the values as strings instead of numerical values.

9. **Editor:** Modify the source code as indicated in bold, then save the file:

```
alert("4 + 9 = " + (x + y));
```

By adding parentheses around the `x + y` section of the expression, you are asking JavaScript to first calculate (`x + y`) and then attach it to the preceding text string.

10. **Browser:** Reload `lab2-5.htm`. Now, as you click **OK,** you should see the alert boxes shown in Figure 2-14 and Figure 2-16, in sequence.



```
4 + 9 = 13
```

Figure 2-16: Correct sum is indicated

You have seen the importance of indicating to the script whether you want to add or concatenate numbers. In this lab, when you included the additional set of parentheses around the `x + y` portion of the statement, JavaScript interpreted that you wanted to perform that operation first before concatenating the additional values.

Inline Scripting, Simple User Events and the *onLoad* and *onUnload* Event Handlers

INSTRUCTOR NOTE: This section demonstrates to the students how to create an alert dialog box that will execute via an **onUnload** event handler.

You have created scripts that exist between the <SCRIPT> </SCRIPT> tags. However, you can also include JavaScript within many of the HTML tags with which you are already familiar. This practice is called inline scripting. For example, you can enclose JavaScript within the <BODY> tag as follows:

```
<BODY onUnload="alert('Goodbye,' + userName + '! Hurry back!');">
```

The `onUnload` event handler intercepts the unloading of a Web page and allows the programmer to associate executable code with this event. Note that the `onUnload` keyword is added to the <BODY> tag as an attribute. This example code sends an alert to users whenever they leave the page. In addition to the simple alert shown here, you can also include a script that refers to more complex functions and variables. For example, you can include a reference to the load event that automatically runs a separate instance of the Web browser.

INSTRUCTOR NOTE: The **onLoad** event handler does not execute associated code until all images, plug-ins and Java applets have downloaded completely to the browser. Attempting to script to an object before it has fully loaded into the browser can cause script errors. Using the **onLoad** event handler ensures that all objects associated with the document are present before any script attempts to access or manipulate them.

This script relies on a user event. Examples of user events include loading and unloading a page, clicking the mouse or pressing a computer key. JavaScript is able to respond to most user-generated events.

As previously mentioned, JavaScript responds to events through pieces of code called event handlers. Do not confuse user-generated events with script-driven event handlers.

If you study the code in the previous example, you will see that it uses the `onUnload` event handler to process, or handle, the unload event. When the page (represented by the <BODY> tag) is unloaded, the `alert()` message will appear. Notice that within the `alert()`, only single quotation marks are used to demarcate text. You use double quotation marks to enclose the entire statement, from before `alert` to after the closing parenthesis (but before the final angle bracket). The use of single quotation marks within the statement is necessary for the JavaScript interpreter to determine the beginning and end of the statement. Note that you could switch the quotation marks as shown in the following example:

```
<BODY onUnload='alert("Goodbye, " + userName + "! Hurry back!");'>
```

This code would produce the same result as the first example. Adopt one quoting scheme or the other. Good coding practice is to adopt one method and use it consistently throughout your JavaScript code.

You can also use the `onLoad` event handler. In fact, JavaScript implies its use; thus far, every script you have written has used the `load` event and `onLoad` event handler because the scripts run immediately as the page is loaded in the browser. You will learn more about events and event handlers later in this course.

INSTRUCTOR NOTE: The concept of keywords as a built-in vocabulary is common to all programming environments. Note that JavaScript's list of keywords and reserved words is similar to Java's keywords. Some of these words may not be functional in JavaScript at this time. Developers need to keep up-to-date with the latest changes in JavaScript as various reserved words gain functionality.

The optional lab for this lesson will provide an opportunity for you to use the `onUnload` event handler.

Keywords and Reserved Words

JavaScript contains keywords that you must use to achieve specific results. JavaScript also contains reserved words that you cannot use for variable or function names.

JavaScript keywords

Keywords are the predefined identifiers that form the foundation of JavaScript. They perform unique duties such as declaring variables (`var`) and defining functions (`function`). You must use correct syntax with these words, including proper spelling and case. Note that some keywords are specific to particular versions of the language. You will learn more about the specific functions of these words in this course. JavaScript keywords include the following:

- `break`
- `catch` (v1.4)
- `const` (v1.5)
- `continue`
- `do` (v1.2)
- `else`
- `export` (v1.2)
- `false`
- `finally` (v1.4)
- `for`
- `function`
- `if`
- `import` (v1.2)
- `in`
- `int`
- `new`
- `null`
- `return`
- `switch` (v1.2)
- `this`
- `throw` (v1.4)
- `true`
- `try` (v1.4)
- `typeof` (v1.1)
- `var`
- `void` (v1.1)
- `while`
- `with`

JavaScript reserved words

You cannot use reserved words as variables, functions, objects or methods. JavaScript reserved words include the preceding keywords as well as the following:

- `abstract`
- `boolean`
- `byte`
- `case`
- `char`
- `class`
- `default`
- `delete`
- `debugger`
- `double`
- `enum`
- `extends`
- `final`
- `float`
- `goto`
- `implements`
- `instanceof`
- `interface`
- `long`
- `native`
- `package`
- `private`
- `protected`
- `public`
- `short`
- `static`
- `synchronized`
- `super`
- `throws`
- `transient`
- `volatile`

Lesson Summary

Application project

These projects will challenge you to use what you have learned in this lesson.

Create an HTML page with a JavaScript block in the `<HEAD>` section of the document and another JavaScript block in the `<BODY>` section. In the first JavaScript block, create four variables named `string1`, `string2`, `string3` and `string4`. Assign the value `start` to `string1`. Assign the return value of a `prompt()` method as the value for `string2`. Assign the return value of a `confirm()` method as the value for `string3`. Assign the value `end` to `string4`.

In the second JavaScript block, create a `document.write()` statement that outputs the text *Hello, World*. Make this text Heading Level 3 (`<H3>`). Create a second `document.write()` that outputs the values of each of the four variables created in the first JavaScript block. Experiment with different methods of ensuring that a space exists between each value. After the code is working properly, determine a way to output the four variable values on separate lines of the HTML page.

As you experiment with the page, observe the values that are returned in various situations. For instance, note the value that is returned if the user selects the Cancel button on a prompt dialog box.

Create a second HTML page with a JavaScript block in the `<HEAD>` section of the document and another JavaScript block in the `<BODY>` section. In the first JavaScript block, create two variables named `number1` and `number2`. Assign different numerical values to each variable. In the second JavaScript block, experiment with the various arithmetic and comparison operators shown in Table 2-3. Create a `document.write()` statement that outputs the results of the operations.

Skills review

The `alert()` method is a simple JavaScript method that allows you to communicate with the user. The `prompt()` method requests user input through a text field within a dialog box. Both `alert()` and `prompt()` are methods of the `window` object. The `confirm()` method is similar to the `alert()` method with one significant exception: The `confirm()` method returns a value of either true or false, whereas the `alert()` method has no return value. The `document.write()` method belongs to the `document` object and allows you to create text that is dynamically written to the window as the script is executed. A variable is a named space of memory: It is a container that holds a value, which you can then access repeatedly in your script. Variables can store five data types in JavaScript: number, string, Boolean, null and Object. In JavaScript, you can declare a variable using the `var` keyword. A JavaScript expression is a part of a statement that is evaluated as a value. JavaScript uses four types of expressions: assignment, arithmetic, string and logical. In JavaScript, operators are used in expressions to store or return a value, generally manipulating operands. You can include JavaScript within many of the HTML tags with which you are already familiar: a practice called inline scripting. JavaScript language contains certain words known as keywords which you must use to achieve specific results. JavaScript also contains reserved words that are intended for specified use. You cannot use keywords or reserved words for variable or function names.

Now that you have completed this lesson, you should be able to:

- ✓ Communicate with users through the `alert()`, `prompt()` and `confirm()` methods.
- ✓ Define variables.
- ✓ Define data types.
- ✓ Obtain user input and store it in variables.
- ✓ Report variable text to the client window.
- ✓ Discern between concatenation and addition.
- ✓ Use expressions.
- ✓ Use operators.
- ✓ Define inline scripting.
- ✓ Implement simple events such as `onLoad()` and `onUnload()`.
- ✓ Define keywords and reserved words.

Lesson 2 Review

1. How does JavaScript allow you to communicate with Web users?

   JavaScript allows you to script actions that occur in response to user events. You can display messages to users, request input from users, and incorporate user input in response messages. JavaScript enables you to communicate in these ways by manipulating objects, and by naming and using variables.

2. Which simple JavaScript methods can you use to communicate with users?

   The `alert()`, `prompt()`, `confirm()` and `document.write()` methods.

3. What event can trigger JavaScript functions as an HTML page loads?

   The `load` event.

4. What is concatenation?

   Synthesis of code to simplify it and reduce duplication. In JavaScript, concatenation is often used to combine text strings for further manipulation.

5. When more than one method is defined in a single line of JavaScript code, which method will execute first?

   A method defined inside another method will always be executed before the outer method.

6. The `alert()`, `prompt()` and `confirm()` methods are methods of which object?

   The `window` object.

7. In programming, which term refers to the syntax used to associate an object's name with its properties or methods?

   Dot notation.

8. What is the purpose of the `document.write()` method?

   To write text into the document dynamically as the page loads in the browser.

9. Name the five data types that can be stored in a variable in JavaScript.

   Object, Boolean, number, string and null.

10. In JavaScript, what is an expression? Name the four JavaScript expressions types.

    An expression is a part of a JavaScript statement that is evaluated as a value or assigns a value to a variable. The four JavaScript expression types are assignment, arithmetic, string and logical.

11. In JavaScript, what is an operator? List at least three operators and describe their functions or types.

    An operator is a character or characters used in an expression to store or return a value, usually manipulating operands (data) in the process. Examples of operators that can be listed for this question include = (assignment), + (arithmetic or string), && (logical), == (comparison), ++ (unary), and so forth. (A complete list of operators is provided in a later lesson.)

12. What is an event handler? Name two simple event handlers.

    An event handler is a JavaScript mechanism for intercepting an event and associating executable code with that event. Two simple event handlers are `onLoad` and `onUnload`.

13. Consider the following JavaScript statements:

```
var myValue = 20;
myValue += 20;
myValue = myValue * 10;
myValue += "20";
myValue = "New value";
```

    For each of these statements, what does the `myValue` variable evaluate to after each statement executes in JavaScript?

    The result would appear as follows:

    20

    40

    400

    40020

    `New value`

Lesson 2 Instructor Section

This section is a supplement containing additional tasks for students to complete in conjunction with the lesson. It also contains additional instructor notes. The instructor may use all, some or none of these additional tools, as appropriate to the specific learning environment. These elements are:

- **Additional Instructor Notes**
  Detailed instructor notes that expand the tips and suggestions presented in the instructor margin notes throughout the lesson.
- **Optional Labs**
  Computer-based labs to be completed during class or as homework.
- **Lesson Quiz**
  Multiple-choice test to assess student knowledge of lesson material.

Additional Instructor Notes

The following section contains detailed instructor notes that expand the tips and suggestions presented in the instructor margin notes throughout the lesson.

Instructor Note 2-1

Location: Using JavaScript to Communicate with the User

This instructor note provides additional information concerning the `alert()` method.

Many students ask whether the text in the title bar of an alert dialog box can be changed. Unfortunately the title bar text cannot be changed or deleted by a script due to security reasons. Not being able to change the text in the title bar helps prevent malicious scripts from misleading users into taking ill-advised actions. If the title of the dialog box could be changed, a malicious script could create a dialog box that appears to be a system dialog box, for example. The user could be prompted to divulge password or other personal information. The security benefits far outweigh any inconvenience caused by static text in the title bar of an alert dialog box.

Instructor Note 2-2

Location: Using JavaScript to Communicate with the User

The example in the **Concatenation** section demonstrates how to embed a `prompt()` method inside an `alert()` method. Students will apply this information in a lab. This example is the students' reference point if they need to see how to perform this task.

Instructor Note 2-3

Location: Using JavaScript to Communicate with the User

The example in the **Requesting confirmation: The *confirm()* method** section demonstrates how to create a confirm dialog box. In Lab 2-3, students will embed a `confirm()` method in an `alert()` method to display the return value from the confirm dialog box. The concept of embedding the `confirm()` method is the same as embedding the `prompt()` method in the `alert()` method. Students will apply this information in the next lab. This example and the example mentioned in Instructor Note 2-2 are the students' reference points if they need to see how to perform this task.

Optional Lab 2-1: Using the JavaScript *onUnload* event handler and inline scripting

In this lab, you will obtain the user's name as he or she enters the page, greet the user by name, write that name on the page to personalize it, then use the name one last time as the page is unloaded. You will use both inline scripting and the `onUnload` event handler.

1. **Editor:** Open **optionalLab2-1.htm** from the Lesson 2 folder of the Student_Files directory.

2. **Editor:** Enter source code in the `<BODY>` tag. Add an **`onUnload`** event handler that calls an `alert()` box. The text should read `Goodbye`, with the `userName` variable concatenated into the string, then the text `Hurry back!`

3. **Editor:** Save **optionalLab2-1.htm**.

4. **Browser:** Open **optionalLab2-1.htm**. You should see a series of dialog and alert boxes. Enter your name in the prompt dialog box and click **OK** as shown in Figure OL2-1.

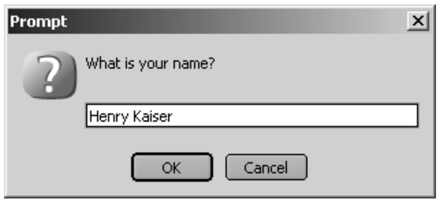
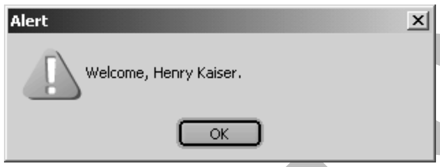


Figure OL2-1: Prompt dialog box

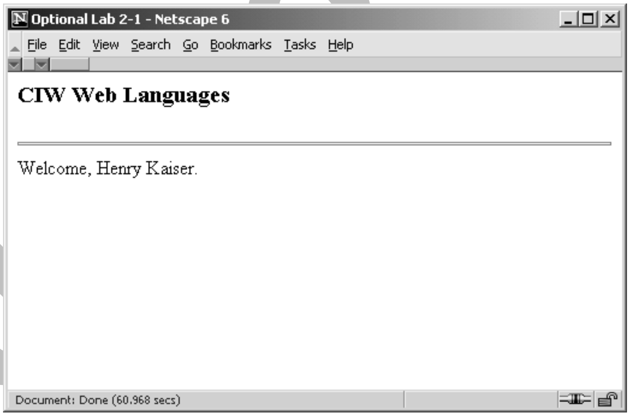
5. **Browser:** The greeting alert box will appear, as shown in Figure OL2-2. Click **OK**.



Welcome, Henry Kaiser.

Figure OL2-2: Alert box

6. **Browser:** The optionalLab2-1.htm page will appear with a personalized greeting, as shown in Figure OL2-3.

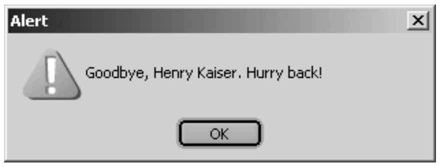

Optional Lab 2-1 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
Document: Done (60.968 secs)

CIW Web Languages

Welcome, Henry Kaiser.

Figure OL2-3: OptionalLab.htm with welcome message

7. **Browser:** Select the **Back** button from your browser's tool bar to navigate to a different page. You should see the alert box shown in Figure OL2-4.



Goodbye, Henry Kaiser. Hurry back!

Figure OL2-4: Goodbye alert box

8. **Browser:** After clicking **OK** on the alert dialog box, your screen will show the page to which you navigated.

This optional lab provided an opportunity to add inline scripting using the `onUnload` event handler. You will learn more about inline scripting, events and event handlers in the coming lessons.

Lesson 2 Quiz

1. Which method is used to request and capture data from a user?

   a. *The `prompt()` method*
   b. The `confirm()` method
   c. The `alert()` method
   d. The `open()` method

2. Which method is used to ask the user a yes-or-no type question?

   a. The `alert()` method
   b. The `open()` method
   c. *The `confirm()` method*
   d. The `prompt()` method

3. Which method is used to send a message to the user?

   a. The `open()` method
   b. *The `alert()` method*
   c. The `prompt()` method
   d. The `confirm()` method

4. In JavaScript, what is a keyword?

   a. A reference used to access a property or method of an object
   b. *A word used to achieve specific results that cannot be used for any other purpose in JavaScript code*
   c. Script within an HTML tag
   d. A named space of memory

5. In JavaScript, what is a reserved word?

   a. A container that holds a value
   b. A character used in an expression to store or return a value
   c. Part of a statement evaluated as a value
   d. *A keyword intended for specified use that cannot be used for any other purpose in JavaScript code*

6. In JavaScript, which of the following items allows you to store user-entered or programmer-defined data and use it more than once?

   a. Keyword
   b. *Variable*
   c. Operator
   d. Concatenation

7. In JavaScript, what is a data type?

   a. A container that holds a value
   b. *A definition of the kind of information a variable holds*
   c. A word used to achieve specific results
   d. A character used in an expression to store or return a value

8. In JavaScript, what is inline scripting?

   a. Part of a statement evaluated as a value
   b. A mechanism for associating code with an event
   c. *Script within an HTML tag*
   d. A reference used to access a property or method of an object

9. Which JavaScript variable data type can represent a series of alphanumeric characters?

   a. Object
   b. *String*
   c. Number
   d. Boolean

10. Which type of JavaScript expression evaluates to true or false?

    a. Assignment
    b. Arithmetic
    c. Boolean
    d. *Logical*

11. Which of the following examples is a valid variable name in JavaScript?

    a. `first&LastName`
    b. `1stName`
    c. *`_lastName`*
    d. `phone#`

12. Which statement uses proper JavaScript syntax for declaring variables?

    a. `var ~correctAnswer = "10";`
    b. `var "userName = Inga";`
    c. *`var firstName = "Sal", lastName = "Fiore";`*
    d. `var username == "Inga";`

13. Which JavaScript operator indicates concatenation?

    a. =
    b. ++
    c. *+*
    d. &&

14. Which JavaScript operator indicates assignment?

    a. *=*
    b. ==
    c. !
    d. =>

15. Which inline-scripted code will implement a simple event that takes place when the user first loads the Web page?

    a. `<BODY onUnload="alert('Hello!');">`
    b. `<SCRIPT LANGUAGE="JavaScript"> alert("Hello!"); </SCRIPT>`
    c. *`<BODY onLoad="alert('Hello!');">`*
    d. `<SCRIPT LANGUAGE="JavaScript"> onLoad="alert('Hello!')"; </SCRIPT>`

Lesson 3: Functions, Methods and Events in JavaScript

Objectives

By the end of this lesson, you will be able to:

- Define functions.
- Call functions.
- Pass arguments to functions.
- Return values from functions.
- Define operator precedence.
- Discern between global and local variables.
- Employ the conditional operator.
- Identify user events and event handlers.
- Use methods as functions.
- Use conversion methods.

Pre-Assessment Questions

1. Which of the following examples demonstrates the correct syntax for a JavaScript function?

   a.
   ```
   Function myFunction() {
     alert("test function");
   }
   ```
   b.
   ```
   function myFunction {
     alert("test function");
   }
   ```
   c.
   ```
   Function myFunction{()
     alert("test function");
   }
   ```
   d.
   ```
   function myFunction() {
     alert("test function");
   }
   ```

2. Which of the following examples demonstrates the correct syntax for passing and receiving parameters for a JavaScript function?

   a.
   ```
   function myFunction(arg1, arg2) {
     alert(arg1 + " " + arg2);
   }

   myFunction("first argument", "second argument");
   ```
   b.
   ```
   function myFunction(arg1, arg2) {
     alert(arg1 + " " + arg2);
   }

   myFunction(first argument, second argument);
   ```
   c.
   ```
   function myFunction("arg1", "arg2") {
     alert(arg1 + " " + arg2);
   }

   myFunction("first argument", "second argument");
   ```
   d.
   ```
   function myFunction(arg1; arg2) {
     alert(arg1 + " " + arg2);
   }

   myFunction("first argument"; "second argument");
   ```

3. Discuss the output of the following JavaScript statements.

   ```
   var x = 4;
   var y = 8;
   var z = 2;
   var result = x + y / z;
   ```

   The `result` *variable would return a value of 8 instead of a value of 6. This is because the division operator (/) takes precedence over the addition operator (+). Thus the* `y / z` *portion of the statement is evaluated first. If the* `x + y` *portion of the statement needed to be evaluated first, the statement would be written as* `var result = (x + y) / z;`*.*

INSTRUCTOR NOTE: JavaScript makes no distinction between a function that returns a value and a function that does not. Some programming languages require different syntax for procedures that simply perform actions, and functions that perform actions and return a value.

Functions

You have learned a few methods of the JavaScript language in this course, including `alert()`, `prompt()`, `confirm()` and `document.write()`. Some methods are also called functions because they can return values. The `document.write()` method, for example, does not return a value to any other variable or function. The `prompt()` method, however, returns a value in the form of a text string, which you can then pass to a variable or to another method, such as `alert()`.

To take full advantage of such values, you must be able to define your own processes, called **functions**. Functions are fundamental tools in JavaScript. By plugging data into a function, a value will return either to the screen, to a variable, or to another function.

function
A named block of code that can be called when needed. In JavaScript, a function can return a value.

A user-defined function is a named, organized block of code that handles actions generated by user events. Functions enable you to write code and place it into a single unit that can perform a specific task repeatedly throughout the program, as needed.

In JavaScript, you can use several built-in functions to improve your program's efficiency and readability. In addition, you can create your own functions to make your programs scalable.

To use functions effectively, you need to know how to declare functions, pass arguments to them, call them, and return function results. After these discussions, this lesson will briefly discuss JavaScript event handlers and built-in JavaScript functions.

Defining a Function

calling statement
A statement that transfers program execution to a subroutine, procedure or function. When the subroutine is complete, execution transfers back to the command following the call statement.

You define a function when you encompass a group of script statements into a function block. Function blocks begin with the keyword `function` followed by the function name. The rest of the statements that form the function must be enclosed in curly brace `{}` characters. A function's statements are processed when the function is called by a **calling statement**. A function can be called by another function or script block. Functions can also be called by events, such as a mouse click, or a page loading or unloading.

The generic syntax for a function is as follows:

```
function functionName(argument1, argument2, ...) {
//statement(s) here
}
```

INSTRUCTOR NOTE: The example in this section demonstrates how to create a user-defined function. Students will apply this information in the next lab. They will not copy code. This example is the students' reference point if they need to see how to perform this task.

The keyword `function` must be typed as shown, in all lowercase letters. The user defines the function name in this example. Function names must be unique for each particular page. As with variable names, function names should reflect the function's intended purpose. Also, the naming rules that apply to variables also apply to function names. The function name must begin with a letter or underscore character; subsequent characters can be letters, numbers and underscore characters.

argument
A value or expression containing data or code that is passed on to a function or procedure.

The curly braces must encompass any statements that are to execute as part of that function. Parentheses () must follow the user-defined function name. These parentheses are used to receive **arguments**, but a function does not necessarily have to take or use them. Arguments (often referred to as parameters) are pieces of data that the function receives and manipulates within the function. A function can receive as many arguments as it needs to perform its task.

One of the most powerful features of a function is the ability to return a value to the calling statement. The `return` keyword is used to return values from a function. This concept will be discussed later in this lesson.

Inserting functions into HTML pages

Define all your functions in the <HEAD> element of your HTML page. That way, when the page is loaded into the browser, the functions will load before any user event can occur that might call the function. Also, your function definitions will be grouped together, which makes troubleshooting easier.

Good coding practice

INSTRUCTOR NOTE: Many references supply information on accepted coding practices, such as "Code Complete" by Steve McConnell. Many online documents also provide good information on this subject. Encourage students to investigate various resources.

In the next lab, you will create your first JavaScript function. Note that programmers have created coding standards and practices that are integral to writing effective, readable code. One of these standards is code indentation. The following JavaScript function exemplifies this concept:

```
<SCRIPT LANGUAGE="JavaScript">
<!--

  function addNumbers (myNumber1, myNumber2) {

      //This function adds two numbers and displays the result

      var x = myNumber1;
      var y = myNumber2;
      var z = x + y;

      alert(z);

  }

//-->
</SCRIPT>
```

This example demonstrates two coding practices. First, a short comment indicates the function's intended purpose. Some programmers prefer to place this type of explanation before the beginning of the function definition, whereas others prefer to place it inside the function as shown in the example. Secondly, note that the statements that compose the function are indented from the function keyword. In this example, the code is indented four spaces. This spacing scheme has no effect on the performance of the function, but makes the code much easier to read. In this example, the benefit of indentation may not be as apparent as it is in a script that contains many more lines. Adopt consistent coding practices as you learn JavaScript.

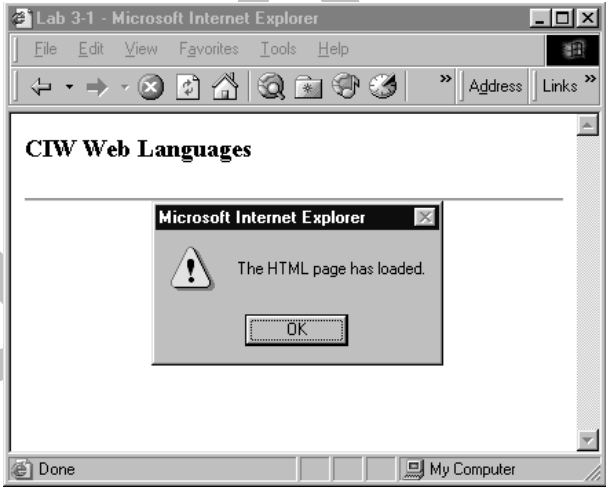
Lab 3-1: Creating a user-defined function in JavaScript

In this lab, you will create a simple function that displays an alert dialog box. The function will be invoked by the `onLoad` event handler.

1. **Editor:** Open **lab3-1.htm** from the Lesson 3 folder of the Student_Files directory.
2. **Editor:** Create a function named **myFunction()** in the <SCRIPT> block of the document's <HEAD> section. After entering the **function** keyword and the name of the function, be sure to properly open the function definition with a curly brace.
3. **Editor:** Inside the function, create an alert dialog box. Use **The HTML page has loaded** for the alert's message.
4. **Editor:** You will now add an `onLoad` event handler to the document's <BODY> tag. The `onLoad` event handler will be used to call the function that you just created. Locate the <BODY> tag near the top of the document. Modify the <BODY> tag as indicated in bold:

   ```
   <BODY onLoad="myFunction();">
   ```

5. **Editor:** Save **lab3-1.htm**.
6. **Browser:** Open **lab3-1.htm**. Your screen should resemble Figure 3-1.


Lab 3-1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
CIW Web Languages
Microsoft Internet Explorer
OK
Done
My Computer

The HTML page has loaded.

Figure 3-1: Lab3-1.htm

7. **Browser:** After clicking **OK**, your screen should render the HTML page seen in Figure 3-1.

In this lab, you successfully created a simple function that displayed an alert dialog box. In addition, you launched your function using the `onLoad` event handler and inline scripting.

Calling a Function

INSTRUCTOR NOTE: Emphasize to students the importance of creating and maintaining a library of functions. Experienced programmers always prefer to use code that is already written as opposed to having to write code from scratch.

Calling a function can be performed from one function to another, from a user event, or from a separate `<SCRIPT>` block. In the previous lab, you called your function by using the `onLoad` event handler as an attribute of the `<BODY>` tag. The `onLoad` event handler intercepted the `load` event and invoked the function.

Following is an example of using one function to call another function:

```
<SCRIPT LANGUAGE="JavaScript">
<!--

  var userName;

  function showName() {
    getName();
     alert(userName);
  }

  function getName() {
    userName = prompt("Please enter your name:","");
  }

// -->
</SCRIPT>

<BODY onLoad="showName();">
```

In this example, the function `showName()` is called using the `onLoad` event handler, as in the previous lab. Note that the first line of the `showName()` function calls the `getName()` function. The `getName()` function would then execute in its entirety before processing of the `showName()` function is completed.

You have now seen two techniques for calling functions. The first was to use an event handler to capture an event and invoke the function. The second was to simply include the name of the function in a `<SCRIPT>` block somewhere in the HTML page.

In a later lab, you will see a function called from within another function. First, however, you need to understand two concepts:

- Passing arguments to functions
- Returning values from functions

Passing arguments to functions

INSTRUCTOR NOTE: The `var` keyword is not used to define and initialize function parameters. The JavaScript interpreter automatically initializes function parameters whenever the function is invoked.

Arguments, or parameters, are values passed into a function from outside the function. These values are then used inside the function and discarded after the function has completed its statements. The following script demonstrates the syntax used to pass arguments into a function:

```
<SCRIPT LANGUAGE="JavaScript">
<!--

  function showName(firstName, lastName) {
    document.write("<B>Your first name is: ");
    document.write(firstName + "<BR>");
    document.write("Your last name is: ");
    document.write(lastName + "</B>");
  }

//-->
```

INSTRUCTOR NOTE: Refer to **Instructor Note 3-1** for more information concerning JavaScript functions and arguments.

```
</SCRIPT>

<BODY>

<SCRIPT LANGUAGE="JavaScript">
<!--

  showName("John", "Doe");

//-->
</SCRIPT>

</BODY>
```

INSTRUCTOR NOTE: This section demonstrates how to pass arguments to a function. Students will apply this information in the next lab.

Note that in the second `<SCRIPT>` block, the values being passed to the `showName()` function are in a comma-delimited list. When the function is called, the literal values `"John"` and `"Doe"` are received into the function via the `firstName` and `lastName` parameters.

This example shows string literals being passed. Variables can be passed as well. Numeric literals and variables would not include the quotation marks as do string literals. If a variable is passed to a function, changing the value of the argument within the function does not change the value of the original variable outside the function. And as mentioned, arguments exist only for the life of the function. As many arguments as needed can be passed into the function.

Returning values from functions

JavaScript functions can return values to the calling statement using the `return` statement. The value to be returned is placed after the `return` keyword. The value that is returned from a function can be assigned to a variable or it can be used in an expression. The following script demonstrates the syntax used to return a value from a function:

```
<SCRIPT LANGUAGE="JavaScript">
<!--

  function returnValue() {
    var myProduct;
    myProduct = document.myForm.input1.value * ➘
    document.myForm.input2.value;
    return myProduct;
  }

//-->
</SCRIPT>

<BODY>

<FORM NAME="myForm">
Multiply <INPUT TYPE="text" NAME="input1">
By       <INPUT TYPE="text" NAME="input2">
<INPUT TYPE="button" VALUE="Result"
onClick=
"alert('The result is ' + returnValue() + '.');">
</FORM>
</BODY>
```

INSTRUCTOR NOTE: This section demonstrates how to return a value from a function to the calling statement. This section also demonstrates how to wrap an `alert()` method around a function call.

In this example, the values entered in the text boxes are multiplied by the `returnValue()` function, with the resulting value returned to the calling statement. This example demonstrates some JavaScript syntax that has not yet been discussed. Consider this line of code:

```
myProduct = document.myForm.input1.value ...
```

This code demonstrates JavaScript syntax for accessing a value entered in an HTML text box and assigning that value to a variable. In this case, one value is multiplied by another and that value is assigned to the `myProduct` variable. Note that the form is named `myForm`, and the text boxes are named `input1` and `input2`. This type of syntax will be discussed in detail later in the course.

Note that the `returnValue()` function is called from within the `alert()` method argument. The name of the function is concatenated into the `alert()` message. Remember that the `return` keyword returns a value to the calling statement. So after the function calculates the numbers, the result (held in `myProduct`) is returned to the calling statement and concatenated into the `alert()` message.

Any type of value can be returned, as can global and local variables. Global and local variables will be discussed following a brief discussion of operator precedence.

Operator precedence

INSTRUCTOR NOTE: In JavaScript, all mathematical and string concatenation operators take precedence over comparison operators, allowing expressions that act as operands for comparisons to be completely evaluated before they are used for comparison operations.

When a JavaScript interpreter reads JavaScript, it must determine which operations to perform first. Operator precedence determines which expressions will be evaluated before others. Consider the expression 4 + 6 * 2, for example. In JavaScript, multiplication takes precedence over addition, and thus the expression would evaluate to 16. However, if you were to rewrite this expression as (4 + 6) * 2, it would evaluate to 20 because parenthetical expressions take precedence over multiplication. See the related appendix for more information about operator precedence.

Global vs. local variables

If you declare a variable within a function definition, the only way you can access or change it is from within that function. Once you declare a variable within a function, the variable is known as a local variable, and you cannot access its value from any other function except the one in which it was declared.

Occasionally, you will want to declare a variable that can be accessed by more than one function or script block in your program. You will need to declare that variable outside of any function definition, usually at the start of your script. This variable is known as a global variable. If you declare a global variable, you can then access its value from any function or script block on that page.

Tech Note: *JavaScript allows you to use the same variable name in different functions. However, same-named variables are treated as separate variables. Note that you should create different names for variables in your scripts whenever possible.*

Recall that JavaScript provides the `var` keyword to declare variables. Recall also that JavaScript does not require the `var` keyword when you want to create a variable. You can simply name the variable in your scripts and start using it, because JavaScript is a loosely typed language. You do not have to declare the data type of a variable before using it, and a variable can have its data type changed over the course of a script.

Be aware that not using the `var` keyword can cause some unexpected behavior in certain situations. As previously stated, a variable created in a function took on local scope and was available only to that script. That statement is true provided that you use the `var` keyword to declare the variable. Any variable that is created without the `var` keyword is global in scope, regardless of where that variable was created. However, a variable created under this circumstance is not available until the function is called. This is another good reason to always use the `var` keyword to declare your variables prior to using them in your scripts.

Lab 3-2: Using functions, arguments and return values in JavaScript

In this lab, you will create a simple function, pass arguments to that function when it is called, and return a value to the calling statement.

1. **Editor:** Open **lab3-2.htm** from the Lesson 3 folder of the Student_Files directory.
2. **Editor:** A simple function has been created for you. The lab3-2.htm file contains a form button. Add an **`onClick`** event handler to the **`<INPUT>`** tag. Use this event handler to invoke **`myFunction()`**.
3. **Editor:** Save **lab3-2.htm**.
4. **Browser:** Open **lab3-2.htm**. Your screen should resemble Figure 3-2.

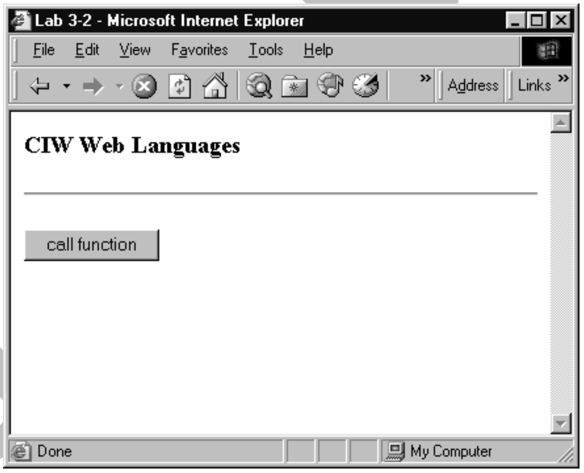

Lab 3-2 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
CIW Web Languages
call function
Done
My Computer

Figure 3-2: Lab3-2.htm

5. **Browser:** Click **call function**. You should see an alert as shown in Figure 3-3.

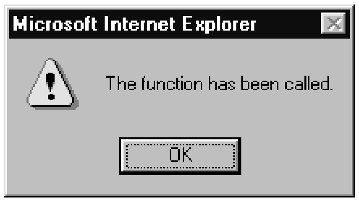


Figure 3-3: Result of function call

6. **Browser:** Click **OK**.

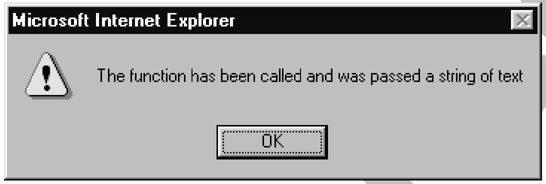
7. **Editor:** Open **lab3-2.1.htm**. You will now add an argument to a function call, then use that argument in the function. Locate the `<INPUT>` tag in the HTML form. Add this string:

   `"a string of text"`

   in the parentheses after the function name in the `onClick` expression. You are passing literal text to the function, so be sure to include the quotation marks around the string.

8. **Editor:** Add an argument named **`arg1`** in the parentheses after the function name. Concatenate `arg1` into the argument of the **`alert()`** method. Save **lab3-2.1.htm**.

9. **Browser:** Open **lab3-2.1.htm**. Your screen should resemble Figure 3-2. Click **call function**. You should see an alert as shown in Figure 3-4.


Microsoft Internet Explorer

The function has been called and was passed a string of text

Figure 3-4: Result of function call

10. **Browser:** Click **OK**.

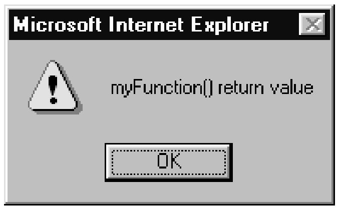
11. **Editor:** Open **lab3-2.2.htm**. Add a `return` statement to `myFunction()` after the `alert()` statement. Add this text:

    `"myFunction() return value"`

    after the `return` keyword.

12. **Editor:** In order to see the return value from the function, you will add an `alert()` method to the calling statement. Locate the `onClick` event handler defined in the `<INPUT>` tag near the bottom of the file. Wrap an `alert()` method around the call to `myFunction()`. Save **lab3-2.2.htm**.

13. **Browser:** Open **lab3-2.2.htm**. Your screen should resemble Figure 3-2. Click **call function**. You should see an alert as shown in Figure 3-4. Click **OK**. You should then see an alert as shown in Figure 3-5.


Microsoft Internet Explorer

myFunction() return value

Figure 3-5: Return value from myFunction()

pass by value
Mode that describes when values are passed to a function, and the function's parameters receive a copy of its argument's value.

pass by reference
Mode that describes when values are passed to a function, and the function's parameters receive its argument's actual value.

In the preceding lab, you created a function that received an argument. When you created the calling statement, you passed a value to the function as an argument. You then returned a value to the calling statement. When a value is passed into a function and changed, the value of the original data is not changed. In programming, this mode is known as **pass by value**. The exception to this rule is if an object reference is passed as an argument to a function. JavaScript uses **pass by reference** for object arguments. For more information concerning pass by reference, see the related appendix.

The next lab will use one function to call another. It will also pass a value to that function, and then return a value to the calling statement. The program is in the form of an online quiz. Each time the user answers a question, the `runnigtotal()` function tabulates the running total of points that user has received for correct answers. The conditional assignment operator will assign a point or not, depending on whether the user's answer to a quiz question is correct. In addition, this lab will use the concept of operator precedence. This lab is rather complex, so be sure to refer to the Lab Recap which follows the lab for a line-by-line analysis.

INSTRUCTOR NOTE: The `runningTotal()` function returns the current value of the `numCorrect` variable to the `takeTest()` function. This value is assigned to the `points` variable, which is local to the `takeTest()` function, which is necessary to demonstrate the passing of values from one function to another. The `numCorrect` variable is global in scope. Since its value is available to both functions, there is no need to return the `numCorrect` variable to the `takeTest()` function unless it is assigned to a local variable inside the function.

Lab 3-3: Calling a function from within another function in JavaScript

1. **Editor:** Open **lab3-3.htm** from the Lesson 3 folder of the Student_Files directory.

2. **Editor:** Examine the following source code:

```
<SCRIPT LANGUAGE="JavaScript">
<!--

var numCorrect = 0;

function takeTest() {
var response = "";
var points = 0;

var q1 = "What company developed JavaScript?";
var a1 = "NETSCAPE";

var q2 = "Using JavaScript operator precedence,\n what is the
result of the following expression? 2 + 4 * 6";
var a2 = 26;

var q3 = "With what object-oriented programming language\n is JavaScript often
compared and confused?";
var a3 = "JAVA";

response = prompt(q1,"");
if (response) points= runningTotal((response.toUpperCase() == a1) ? 1 : 0);
alert(points);

response = prompt(q2,"");
if(response) points= runningTotal((response == a2) ? 1 : 0);
alert(points);

response = prompt(q3,"");
if (response) points=runningTotal((response.toUpperCase() == a3) ? 1 : 0);
alert("You answered a total of " + points + " correctly.");

numCorrect = 0;
points = 0;

}
```

```
function runningTotal(i) {
    numCorrect += i;
    return numCorrect;
}

// -->
</SCRIPT>
```

3. **Editor**: Locate the `<FORM>` tag near the bottom of the document. Examine the `<INPUT>` tag. Note especially the code indicated in bold:

```
<FORM>
<INPUT TYPE="button" VALUE="take quiz" onClick="takeTest();">
</FORM>
```

4. **Editor:** Close **lab3-3.htm**.

5. **Browser:** Open **lab3-3.htm**. Your screen should resemble Figure 3-6.

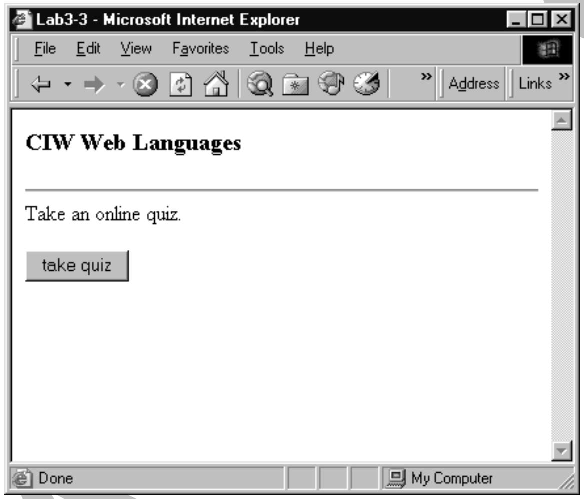


Figure 3-6: Lab3-3.htm

6. **Browser:** Click **take quiz**. Respond to each of the prompts. Enter both correct and incorrect answers to check the logic of the quiz.

Lab recap

INSTRUCTOR NOTE: The values for the q2 and q3 variables use a \n character. This character has not yet been introduced in this course. The \n is a newline character. This character will be discussed in Lesson 6.

Note the use of the `if (response)` statement in the lab code, which is included at key points so that the JavaScript interpreter would not return an error if the user clicked the Cancel option repeatedly. For now, take the time to examine the most relevant parts of this lab in detail. Following are the key script elements, numbered for discussion purposes only.

```
01.  var numCorrect = 0;
02.  function takeTest() {
03.  var response = "";
04.  var points = 0;
05.  var q1 = "What company developed JavaScript?";
06.  var a1 = "NETSCAPE";
07. var q2 = "Using JavaScript operator precedence,\n what is the result
    of the following expression? 2 + 4 * 6";
08. var a2 = 26;
09. var q3 = With what object-oriented programming language\n is
    JavaScript often compared and confused??";
10. var a3 = "JAVA";
11. response = prompt(q1,"");
12. if (response) points = runningTotal((response.toUpperCase() == a1) ?
    1 : 0);
13. alert(points);
14. response = prompt(q2,"");
15. if (response) points= runningTotal((response == a2) ? 1 : 0);
16. alert(points)
17. response = prompt(q3,"")
18. if (response) points = runningTotal((response.toUpperCase()  == a3)
    ? 1 : 0);
19. alert("You answered a total of " + points + " correctly.");
20. numCorrect = 0;
21. points = 0;
22. }
23. function runningTotal(i) {
24. numCorrect += i;
25. return numCorrect;
26. }
```

- **Line 01: Global variable**
  The `numCorrect` variable is declared outside either function, making it a global variable with global scope. This variable is needed by both the `runningTotal()` and the `takeTest()` functions and therefore is defined outside of either function.
- **Line 02: Defining the `takeTest()` function**
  Note the standard syntax of `function takeTest() {` to open the function definition.
- **Lines 03 through 10: Local variables**
  The variables `response` and `points`, and the question and answer variables (`q1`, `a1`, etc.), are declared here within the `takeTest()` function. Their values can only be changed and accessed from within this function, which is appropriate because these variables are not needed in the `runningTotal()` function.
- **Line 11: Collecting the user's answer to a question**
  The variable `response` is assigned the user's input to the `prompt()`. The prompt itself uses the variable `q1`, which contains the text of the first question.

- **Line 12: Comparing the user's answer to the correct answer, and calling the `runningTotal()` function**
  This line reads as follows:

  ```
  if (response) points= runningTotal((response.toUpperCase() ➘
  == a1) ? 1 : 0)
  ```

  JavaScript will process this expression in the following order.

  First: `if (response)`

  An `if` statement checks whether the user has clicked the Cancel button on the prompt dialog box. If so, a null value will be returned. Later in the program, the `toUpperCase()` method will be applied to two of the user's answers. If you attempt to apply the `toUpperCase()` method to a null value, an error will be generated. Thus, the `if` statement averts this possible error message.

  Second: (response.toUpperCase() == a1)

  A logical expression has been presented, and evaluates to true or false, depending on the user's answer.

  Third: JavaScript evaluates (response.toUpperCase() == a1) ? 1 : 0

  If the user's `response` is correct (the expression is true), the value of `1` will be made ready for whatever comes next. If not, the value of `0` will be made ready. The conditional operator `? :` decides which of these values will be returned.

  INSTRUCTOR NOTE: Note that the user's input is compared to the value held by the `a1` variable. Remind students that the answer variables were defined in uppercase letters. Thus the toUpperCase() function is needed to convert the user's response to uppercase letters as well.

  Fourth: The original line will be read by JavaScript as one of the following:

  ```
  runningTotal(1)
  ```

  or:

  ```
  runningTotal(0)
  ```

  JavaScript will at this point call the `runningTotal()` function and at the same time pass one of two values, either `0` or `1`.

At this point, skip forward to Line 23.

- **Lines 23 through 26: The `runningTotal(i)` function**
  Note that the function definition takes an argument named `i`, which has no meaning of its own and is simply a variable that holds whatever value is passed to this function. You know from Line 11 that the value being passed can only be `1` or `0`, and whichever it is will be added to the `numCorrect` variable. If the answer is incorrect, `numCorrect` will be incremented by 0, meaning that its value will remain unchanged. If the answer is correct, `numCorrect` will increase by the value of 1.

  Once `numCorrect` has been manipulated, the current value of `numCorrect` is returned to the line in which the function was called, in this case Line 12. At this point, the return value contained in `numCorrect` is assigned to the `points` variable.

- **Line 13: The running total is displayed**
  The updated `points` value is displayed to the user through the `alert()` method.

- **Lines 14 through 19: Repeating the process**
  With the exception of one variation, these lines basically repeat the process outlined in Lines 11 through 13. The difference is in Line 15. Here, the `toUpperCase()` method is not needed because the answer is an integer.
- **Line 19: The final score**
  This line is a variation on Lines 13 and 16. Because this line follows the last question, code has been added to indicate that the quiz is over.
- **Lines 20 through 21: Resetting a variable**
  The `numCorrect` and `points` variables are reset to `0` in anticipation of the next quiz.
- **Lines 22 and 26: Closing the functions**
  It is easy to forget to close a function. Because every function opens with a curly brace, every function must end with one as well.

You now have seen another example of functions and how they can be used. You have learned that you can call a function on a user event or from another function. You have also seen another example of an argument passed to a function and a value returned to the calling statement.

User Events and JavaScript Event Handlers

Whether they are submitting a form or navigating a Web page, users generate events. JavaScript contains predetermined event handlers that deal with these events. Tables 3-1 and 3-2 list JavaScript events and the event handlers designed to process them. Note that these are commonly used events and event handlers. Many others exist but are beyond the scope of this course.

Table 3-1: JavaScript user events

Event	Description
`abort`	Occurs when the loading of an image is aborted
`blur`	Occurs when input focus is removed from a form element (e.g., when a user clicks the mouse button outside of a particular field)
`click`	Occurs when the user clicks on a link or form element
`change`	Occurs when a user changes the value of a form field
`error`	Occurs when an error takes place while a page or image is loading
`focus`	Occurs when a user gives input or focus to a form element
`load`	Occurs when a page is loaded into the browser
`mouseOver`	Occurs when the user moves the mouse pointer over a link or area object
`mouseOut`	Occurs when the mouse pointer leaves a link or area object
`reset`	Occurs when a form's Reset button is clicked
`select`	Occurs when the user selects the text in a form field
`submit`	Occurs when a form's Submit button is clicked
`unLoad`	Occurs when a page is unloaded from the browser

Table 3-2: JavaScript event handlers

Object	Event Handler(s)
Button	onClick
Reset	onClick
Submit	onClick
Radio	onClick, onBlur, onFocus
Checkbox	onClick, onBlur, onFocus
link	onClick, onMouseOver, onMouseOut
form	onSubmit, onReset
text	onChange, onFocus, onBlur, onSelect
textarea	onChange, onFocus, onBlur, onSelect
select	onChange, onFocus, onBlur
image	onAbort, onError, onLoad
area	onClick, onMouseOver, onMouseOut
window	onLoad, onUnload, onError

INSTRUCTOR NOTE: Almost every object in an HTML document is capable of receiving some type of event. The objects will only respond to these events if the programmer adds event handlers in the appropriate places in document's code. As the students gain more experience with JavaScript, they will see that learning how and when to use event handlers is one of the most important JavaScript skills they can acquire.

As noted earlier, event handlers are usually placed inline with HTML tags. A previous example used the `onUnload` event handler inside a `<BODY>` tag.

You will use many of these events and event handlers elsewhere in this course. An optional lab for this lesson will also provide an opportunity to use JavaScript event handlers.

Methods as Functions

In JavaScript, you can use built-in functions to manipulate object properties or to perform calculations. In some programming languages, methods and functions are two different operations. However, methods and functions are interchangeable in JavaScript. The most helpful distinction between them is that any method that returns a value can be called a function. JavaScript has built-in functions (or methods) and functions that users can define. You have already seen the syntax required to create user-defined functions.

In the following example, the `toUpperCase()` method works as a function because it returns the answer variable after converting it to uppercase letters:

```
var answer;
var correctAnswer = "WASHINGTON, D.C.";
answer = prompt("What is the capital of the United States?","");
answer = answer.toUpperCase();
```

In this code, the user might enter "Washington, D.C." in response to the prompt. But the `correctAnswer` variable contains `"WASHINGTON, D.C."` Because JavaScript is case-sensitive, the user's answer could be read as insufficient simply because the user did not type the answer in capital letters. By using the `toUpperCase()` method, you can convert the answer to its uppercase equivalent, thus making a useful comparison possible. The `toUpperCase()` method operates as a function because it returns a value you can subsequently use in the rest of the program. The previous lab demonstrated the use of the `String` object's `toUpperCase()` method.

Several built-in methods are available in JavaScript that do not belong to any particular object. They are available for use as needed throughout your scripts. Some commonly used examples are the `parseInt()` and `parseFloat()` methods. These two methods will be discussed later in this lesson. Another example is the `isNaN()` method, used to determine whether an operand is a number. These types of functions are referred to as top-level functions. As you learn more about JavaScript, you will use these top-level built-in functions to enhance your scripts.

Using built-in functions

As mentioned, JavaScript has several built-in functions that can make your work easier. Two are handy in mathematical calculations: `parseInt()` and `parseFloat()`.

- `parseInt()` converts a string to its integer equivalent.
- `parseFloat()` converts a string to its **floating-point** decimal equivalent.

floating-point calculation
A calculation in which the decimal point can move as needed to account for significant digits.

Using the `parseInt()` or `parseFloat()` functions in your coding will eliminate the problem of the + operator being used for string concatenation when you want it to be used in a mathematical expression. You can incorporate some of these ideas in an optional lab for this lesson. You will see that using either the `parseInt()` or the `parseFloat()` function solves the problem of JavaScript misinterpreting the dual-purpose + operator.

INSTRUCTOR NOTE: This section demonstrates to the students how to use the `parseInt()` and `parseFloat()` conversion functions.

The following example demonstrates the use of the `parseInt()` and `parseFloat()` functions:

```
<SCRIPT LANGUAGE="JavaScript">
<!--

  var myVar1 = "3 abc";
  var myVar2 = 4.75;
  var myVar3 = "4.75 abc";

  myVar1 = parseInt(myVar1);
  myVar2 = parseInt(myVar2);
  myVar3 = parseFloat(myVar3);

  alert(myVar1); //Displays 3
  alert(myVar2); //Displays 4
  alert(myVar3); //Displays 4.75

//-->
</SCRIPT>
```

In this example, the string `"3 abc"` is converted to the integer `3` and the floating-point number `4.75` is converted to the integer `4` using `parseInt()`. The string `"4.75 abc"` is converted to the floating-point number `4.75` using `parseFloat()`.

Lesson Summary

Application project

This project will challenge you to use what you have learned in this lesson. Create an HTML page with a JavaScript block in the `<HEAD>` section of the document and another JavaScript block in the `<BODY>` section. In the first JavaScript block, create four functions named `add()`, `subtract()`, `multiply()` and `divide()`. Define two arguments for each function, `arg1` and `arg2`, in the parentheses following the function name. Define one line of code in each function, a `return` statement that returns the result of adding, subtracting, multiplying or dividing `arg1` with `arg2`.

In the second JavaScript block, create four `document.write()` statements that invoke each function. Pass two literal values as the arguments for each function. The result of the functions should be displayed on the HTML page.

Skills review

A powerful feature of JavaScript is the fact that programmers can create their own functions. A function is an organized, named block of code that handles actions generated by user events. Giving a block of code a name allows it to be called repeatedly as needed throughout a program. JavaScript functions can receive arguments and return values. JavaScript also provides built-in functions that perform tasks. Calling a function can be performed from one function to another, from a user event, or from a separate script block. A variable declared within a function with the `var` keyword is known as a local variable, and you cannot access its value from any other function except the one in which it was declared. Global variables are declared outside of any function definition, and are available to all functions and script blocks on the page. JavaScript contains predetermined event handlers that process user-generated events. Most objects in JavaScript have events and event handlers associated with them.

Now that you have completed this lesson, you should be able to:

- ✓ Define functions.
- ✓ Call functions.
- ✓ Pass arguments to functions.
- ✓ Return values from functions.
- ✓ Define operator precedence.
- ✓ Discern between global and local variables.
- ✓ Employ the conditional operator.
- ✓ Identify user events and event handlers.
- ✓ Use methods as functions.
- ✓ Use conversion methods.

Lesson 3 Review

1. In programming, what is a function? In JavaScript, how does a function differ from a method?

   A function is a named block of code that can be called when needed. In JavaScript, functions and methods are interchangeable. Any method that returns a value can be called a function.

2. What is a built-in function? Name at least one built-in JavaScript function.

   A built-in function is a value-returning method provided by the JavaScript language with a predefined purpose. JavaScript also allows users to define their own functions. Built-in JavaScript functions include `toUpperCase()`, `isNaN()`, `parseInt()`, `parseFloat()`, and others.

3. How can you define a function in JavaScript?

   You can define your own JavaScript function by encompassing a group of statements in a statement block beginning with the keyword `function`. All statements associated with the function are enclosed within curly braces ({ }).

4. What is a floating-point calculation?

   A calculation in which the decimal point is allowed to move as needed to account for significant digits.

5. Explain good coding practice and when it is required in JavaScript.

   Good coding practice refers to coding standards that are not required but have been developed by programmers to make code more easily readable and effective. Examples include hierarchical code indentation, consistent capitalization schemes, and comments that state a function's intended purpose.

6. In what ways can you call a function in JavaScript?

   You can call a JavaScript function from another function, from a user or browser event, or from a separate <SCRIPT> block on the HTML page.

7. When passing arguments in JavaScript, what must you include with a string literal that is not required for numeric literals?

   Quotation marks surrounding the string.

8. In JavaScript, how many arguments can you pass, and how long does each argument exist?

   You can pass as many arguments as needed into a function. An argument exists only for the life of the function.

9. What is operator precedence? Give at least one example that relates to JavaScript.
   Operator precedence specifies which expressions will be evaluated before others in JavaScript statements. For example, multiplication takes precedence over addition, but parenthetical operators can be used to change operator precedence.

10. How do global variables differ from local variables?
    Local variables are declared within a function and can be accessed only from that function. Global variables are declared outside of any function and can be accessed from any function or <SCRIPT> block you declare.

11. What is pass by value? What is pass by reference? Which does JavaScript use?
    Pass by value means that when values are passed to a function, the function's parameters receive a copy of its argument's value (the value of the original data is not changed). Pass by reference means that when values are passed to a function's parameters, the parameters receive the argument's actual value. JavaScript uses pass by value, with the exception that it uses pass by reference for object arguments.

12. What is the conditional operator, and what is its purpose?
    The conditional operator is indicated as `(condition) ? x : y` (where `(condition)` is a Boolean expression and `x` and `y` are values). The conditional operator is used to decide which of these values will be returned. If the Boolean expression is true, `x` is returned. Otherwise, `y` is returned.

13. What is a user event? Name at least three examples of JavaScript user events.
    A user event is an action performed by a user that can be intercepted by JavaScript and used to call an action. JavaScript user events include `click`, `blur`, `focus`, `load`, `mouseOver`, `submit`, `unLoad` and others.

14. What is an event handler? Name at least three examples of JavaScript event handlers.
    An event handler is a JavaScript mechanism for intercepting an event and associating executable code with that event. JavaScript event handlers include `onClick`, `onBlur`, `onFocus`, `onLoad`, `onMouseOver`, `onSubmit`, `onUnLoad` and others.

Lesson 3 Instructor Section

This section is a supplement containing additional tasks for students to complete in conjunction with the lesson. It also contains additional instructor notes. The instructor may use all, some or none of these additional tools, as appropriate to the specific learning environment. These elements are:

- **Additional Instructor Notes**
  Detailed instructor notes that expand the tips and suggestions presented in the instructor margin notes throughout the lesson.
- **Optional Labs**
  Computer-based labs to be completed during class or as homework.
- **Lesson Quiz**
  Multiple-choice test to assess student knowledge of lesson material.

Additional Instructor Notes

The following section contains detailed instructor notes that expand the tips and suggestions presented in the instructor margin notes throughout the lesson.

Instructor Note 3-1

Location: Calling a Function

Functions in JavaScript can receive a varying number of arguments. Also, explicit placeholders need not be defined in the function signature to use these arguments. As an example, consider the following HTML form:

```
<FORM>
<INPUT TYPE="radio" NAME="listType" checked> Ordered list<BR>
<INPUT TYPE="radio" NAME="listType"> Unordered list<P>
Add argument: <INPUT TYPE="text" NAME="anotherArg"><BR>
Add argument: <INPUT TYPE="text" NAME="anotherArg2"><P>
<INPUT TYPE="button" VALUE="Call function"
 onClick=
"var listStyle = document.myForm.listType[0].checked ? 'o' : 'u';
var anotherArg = document.myForm.anotherArg.value;
var anotherArg2 = document.myForm.anotherArg2.value;
createList(listStyle,'one','two','three',anotherArg, ➘
           AnotherArg2);">
</FORM>
```

This form contains two radio buttons. It also contains two text boxes. If the first radio button is checked, the letter *o* is passed as an argument to the `createList()` function. If the second radio button is checked, the letter *u* is passed to the function. The `liststyle` variable will hold one of these two values. The user is able to enter two more arguments in the text boxes. The appropriate variables receive their values in the `onClick` code. Then the `createList()` function is called, passing the `listStyle` variable, as well as three other arguments and the arguments defined by user input. Consider the `createList()` function:

```
function createList(x) {
  var argLen = createList.arguments.length;
  newWin=open("","argWin","height=220,width=275");
  newWin.document.open();

  newWin.document.write("<HTML><HEAD><TITLE>JavaScript ➘
Functions</TITLE></HEAD>");

  // Start list
  newWin.document.write("<BODY><" + x + "l>");

  // loop through arguments
  for (var i = 0; i < argLen; i++) {
    newWin.document.write("<LI>" + createList.arguments[i]);
  }

  // End list
  newWin.document.write("</" + x + "l><P>");

  newWin.document.write("Total arguments received: " ➘
                       + argLen + "<P>");

  newWin.document.write("</BODY</HTML>");

  newWin.document.close();
}
```

The `createList()` function is scripted to receive just one argument, held in the variable `x` placeholder. Note that the function has an `arguments` property. The `arguments` property has a `length` property that can be used to determine how many arguments have been passed to the function. Note that the *o* or the *u* passed to the function is used to determine whether an `<OL>` or a `<UL>` tag is created. A `for` loop is used to loop through and display the function's arguments. Figure IN3-1 shows this code loaded in a browser.

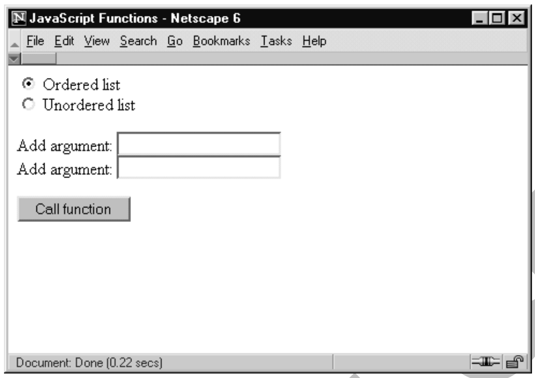
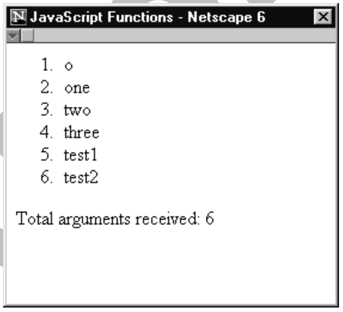

JavaScript Functions - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
Ordered list
Unordered list
Add argument:
Add argument:
Call function
Document: Done (0.22 secs)

Figure IN3-1: Variable arguments demonstration

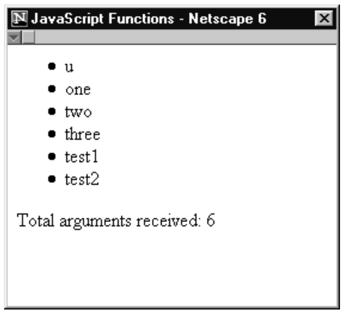
The result of clicking the **Call function** button with the Ordered list radio button checked is shown in Figure IN3-2.



```
1. o
2. one
3. two
4. three
5. test1
6. test2

Total arguments received: 6
```

Figure IN3-2: Result of function call

The result of clicking the **Call function** button with the Unordered list radio button checked is shown in Figure IN3-3.



```
• u
• one
• two
• three
• test1
• test2

Total arguments received: 6
```

Figure IN3-3: Result of function call

This example uses some concepts that have yet to be discussed in the course. This example is provided so that the instructor is aware of this JavaScript concept. It may be more beneficial to introduce this particular example later in the course.

Optional Lab 3-1: Using a JavaScript conversion function

This optional lab will provide an opportunity to use the JavaScript `parseFloat()` conversion function.

1. **Editor:** Open **optionalLab3-1.htm** from the Lesson 3 folder of the Student_Files directory.

2. **Editor:** In the first <SCRIPT> block, examine the `addNumbers()` function that has been defined for you.

```
function addNumbers (arg1, arg2) {
  var result = arg1 + arg2;
  return result;
}
```

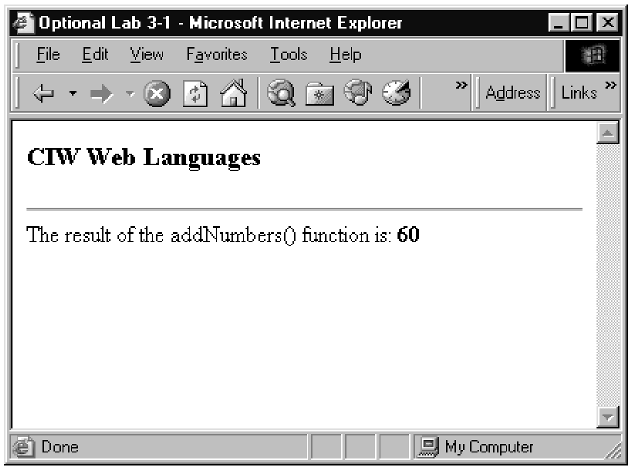
3. **Editor:** This simple function receives two values, adds them together and returns the result. In the second <SCRIPT> block, examine the following code:

```
alert("Please enter a numerical value in each of the ➘
following prompt dialog boxes. \nThe two numbers will ➘
be added together.");

var num1 = prompt("Please enter a numerical value.","");
var num2 = prompt("Please enter a numerical value.","");

document.write("The result of the addNumbers() ➘
function is: <B>" + addNumbers(num1, num2) + "</B><P>");
```

4. **Editor:** Recall that the return value of a `prompt()` method is a string. Determine the various places in the code to use the `parseFloat()` function to convert the user's input to a numerical value. The `parseFloat()` function can be used in several places to achieve the desired result.

5. **Editor:** Save **optionalLab3-1.htm**.

6. **Browser:** Open **optionalLab3-1.htm**. You will see an alert dialog box informing the user of the purpose of the page. You will then see two prompt dialog boxes asking for user input. After you supply the input, your screen should resemble Figure OL3-1, depending on the data input.


Optional Lab 3-1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
Done
My Computer

CIW Web Languages

The result of the addNumbers() function is: **60**

Figure OL3-1: OptionalLab3-1.htm

7. **Browser:** Test the page to ensure that the `addNumbers()` function returns the proper result.

This lab provided an opportunity to use a JavaScript conversion function. You saw that the `parseFloat()` function is used to convert string values to numerical values.

Optional Lab 3-2: Using JavaScript event handlers

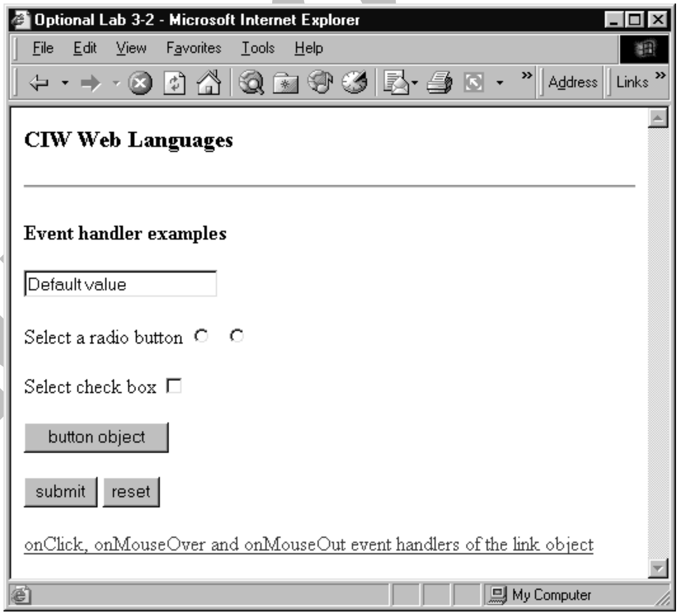
This optional lab will provide an opportunity to use JavaScript event handlers.

1. **Editor:** Open **optionalLab3-2.htm** from the Lesson 3 folder of the Student_Files directory.

2. **Editor:** Locate the HTML comments placed before each HTML object such as the following comment: **`<!-- Add onSubmit and onReset event handlers of the form object -->`** Add the appropriate event handler inline with the appropriate HTML tags. In cases in which more that one event handler is mentioned, add the event handlers in the order in which they are listed in the HTML comment.

3. **Editor:** Save **optionalLab3-2.htm**.

4. **Browser:** Open **optionalLab3-2.htm**. Your screen should resemble Figure OL3-2.


Optional Lab 3-2 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
My Computer

CIW Web Languages

Event handler examples

Default value

Select a radio button

Select check box

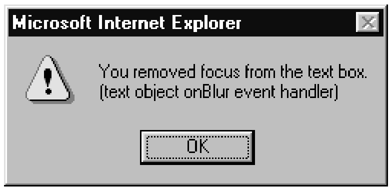
button object

submit reset

onClick, onMouseOver and onMouseOut event handlers of the link object

Figure OL3-2: OptionalLab3-2.htm

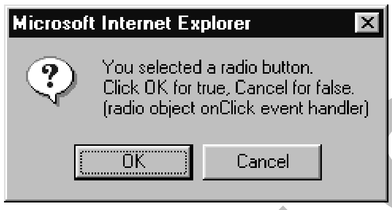
5. **Browser:** Click inside the text box. Removing the cursor from the text box should result in an alert dialog box as shown in Figure OL3-3.


Microsoft Internet Explorer

You removed focus from the text box.
(text object onBlur event handler)

Figure OL3-3: Alert dialog box

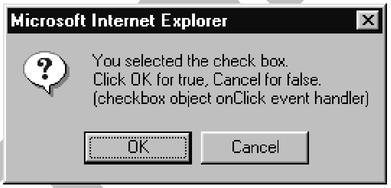
6. **Browser:** Clicking a radio button should result in a confirm dialog box as shown in Figure OL3-4.


Microsoft Internet Explorer

You selected a radio button.
Click OK for true, Cancel for false.
(radio object onClick event handler)

Figure OL3-4: Confirm dialog box

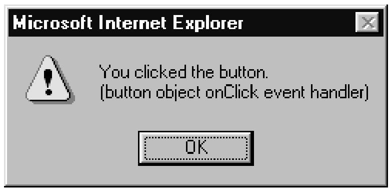
7. **Browser:** Clicking the check box should result in a confirm dialog box as shown in Figure OL3-5.


Microsoft Internet Explorer

You selected the check box.
Click OK for true, Cancel for false.
(checkbox object onClick event handler)

Figure OL3-5: Confirm dialog box

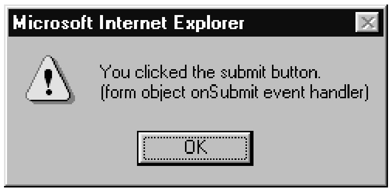
8. **Browser:** Clicking the **button object** button should result in an alert dialog box as shown in Figure OL3-6.


Microsoft Internet Explorer

You clicked the button.
(button object onClick event handler)

Figure OL3-6: Alert dialog box

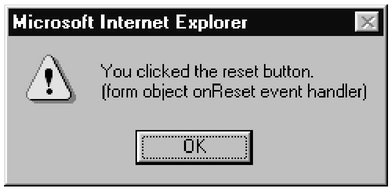
9. **Browser:** Clicking the **submit** button should result in an alert dialog box as shown in Figure OL3-7.


Microsoft Internet Explorer

You clicked the submit button.
(form object onSubmit event handler)

Figure OL3-7: Alert dialog box

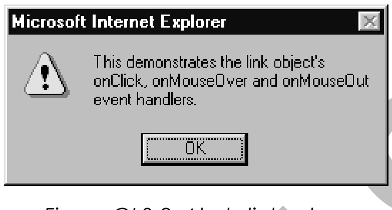
10. **Browser:** Clicking the **reset** button should result in an alert dialog box as shown in Figure OL3-8.


Microsoft Internet Explorer

You clicked the reset button.
(form object onReset event handler)

Figure OL3-8: Alert dialog box

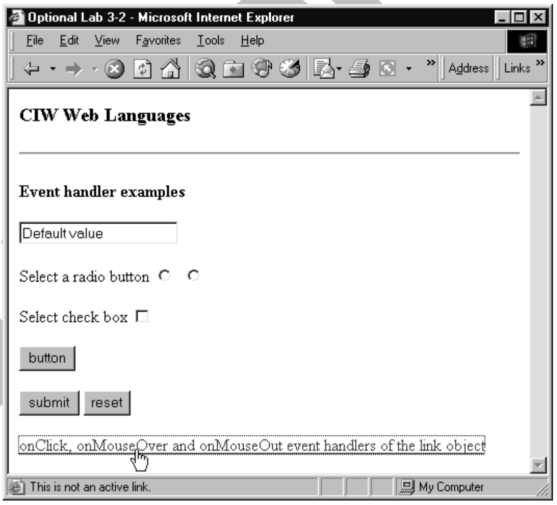
11. **Browser:** Clicking the link should result in an alert dialog box as shown in Figure OL3-9.


Microsoft Internet Explorer

This demonstrates the link object's onClick, onMouseOver and onMouseOut event handlers.

Figure OL3-9: Alert dialog box

12. **Browser:** Placing the cursor over the link should result in a message in the browser's status bar as shown in Figure OL3-10.


Optional Lab 3-2 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links

CIW Web Languages

Event handler examples

Default value

Select a radio button

Select check box

onClick, onMouseOver and onMouseOut event handlers of the link object

This is not an active link.

Figure OL3-10: Message in status bar

INSTRUCTOR NOTE:
OptionalLab3-2.htm includes code that manipulates the `status` property of the `window` object. This concept will be discussed in detail in Lesson 5.

13. **Browser:** Removing the cursor from the link should cause the message displayed in the status bar of Figure OL3-10 to disappear.

14. **Browser:** Experiment with the various objects on the page. In particular, note the behavior of the radio buttons, check boxes and confirm dialog boxes that are launched when these objects are selected.

This lab provided an opportunity to use and observe the behavior of commonly used JavaScript event handlers. Several concepts were introduced in this lab that have yet to be discussed in this course. In particular, you saw the `status` property of the `window` object manipulated with inline script. This syntax will be discussed in detail in another lesson.

Lesson 3 Quiz

1. In JavaScript, what is a function?

   a. An expression containing data
   b. *A named block of code that can be called as needed*
   c. An action performed by an object that returns no value
   d. A statement that transfers program execution to a subroutine

2. The JavaScript `toUpperCase()` method is an example of:

   a. an argument.
   b. a calling statement.
   c. *a built-in function.*
   d. a user-defined function.

3. Which of the following examples is an example of a user-defined JavaScript function?

   a. `parseInt()`
   b. *`function addSpace() {}`*
   c. `isNaN()`
   d. `parseFloat()`

4. Which of the following examples is an example of a JavaScript conversion method?

   a. `document.write()`
   b. *`parseFloat()`*
   c. `alert()`
   d. `function addSpace()`

5. In JavaScript, what is an argument?

   a. Any method that returns a value
   b. A named block of code that can be called
   c. *A value passed into a function from outside the function*
   d. A statement that transfers program execution to a subroutine

6. In JavaScript, the `return` keyword is used to:

   a. *return a value to a function's calling statement.*
   b. return a value to a function.
   c. return a variable to an expression.
   d. return a function to an argument.

7. How does a global variable differ from local variable?

   a. *You can access a global variable value from any function or `<SCRIPT>` block you define on the HTML page.*
   b. You declare a local variable outside of any function.
   c. You declare a global variable within a function.
   d. You can access a local variable value from any function or *`<SCRIPT>`* block you define on the HTML page.

8. Which JavaScript event occurs when the user highlights the text in a form field?

   a. `blur`
   b. *`select`*
   c. `mouseOver`
   d. `click`

9. Which JavaScript event occurs when a Web page is accessed and appears in the browser?

   a. `submit`
   b. `focus`
   c. `change`
   d. *`load`*

10. Which JavaScript event handler can respond to the `checkbox` object?

    a. *`onBlur`*
    b. `onSubmit`
    c. `onMouseOver`
    d. `onSelect`

11. Which JavaScript event handler can respond to the `link` object?

    a. `onLoad`
    b. *`onMouseOut`*
    c. `onFocus`
    d. `onError`

12. Consider the following JavaScript statements:

```
function myTest (myValue) {
  if (myValue < 5) {
    return true;
  }else{
    return;
  }
}
document.write(myTest(6));
```

What is the result of these JavaScript statements? Why?

When called, the `myTest()` function would return `undefined` because nothing follows the keyword `return` in the `else` clause of the `if...else` statement.

13. Consider the following JavaScript statements:

```
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!--

function myFunction() {
  myValue = 10;
  return myValue * 10;
}

//-->
</SCRIPT>

</HEAD>
<BODY>

<SCRIPT LANGUAGE="JavaScript">
<!--

document.write(myValue + "<BR>");
document.write(myFunction());

//-->
</SCRIPT>
</BODY>
```

What is the result of these JavaScript statements? Why?

These statements result in the error message "myValue is not defined." The variable `myValue` is defined without the `var` keyword, making it global in scope. It would seem that the `myValue` variable could be accessed from any script block. However, because `myFunction` is not called until the next line, the `myValue` variable is not accessible to the preceding line. If the second <SCRIPT> block were written as follows, the `myValue` variable would be accessible:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
document.write(myFunction() + "<BR>");
document.write(myValue);

//-->
</SCRIPT>
```

The result of the preceding statements would be as follows:

```
100
10
```

Lesson 4: Controlling Program Flow in JavaScript

Objectives

By the end of this lesson, you will be able to:

- Use the `if...` statement.
- Use the `while...` statement.
- Use the `for...` statement.
- Use the `break` and `continue` statements.
- Define the `do...while` statement.
- Use the `switch...` statement.

Pre-Assessment Questions

1. Which of the following examples demonstrates correct syntax for a JavaScript `if...else if...else` statement?

a.
```
if myCondition == true {
  //statements to execute
}
else if anotherCondition == true {
  //statements to execute
}
else {
  //default statements to execute
}
```

b.
```
if(myCondition = true) {
  //statements to execute
}
else if (anotherCondition = true) {
  //statements to execute
}
else {
  //default statements to execute
}
```

c.
```
if (myCondition == true) {
  //statements to execute
}
else if (anotherCondition == true) {
  //statements to execute
}
else {
  //default statements to execute
}
```

d.
```
if { myCondition = true
  //statements to execute
}
else if { anotherCondition = true
  //statements to execute
}
else {
  //default statements to execute
}
```

2. Which of the following examples demonstrates correct syntax for a JavaScript `switch` statement?

   a.
   ```
   var myVariable = "test string";

   switch myVariable {
     case first string :
     //statements to execute
     break;

     case second string :
     //statements to execute
     break;

     case third string :
     //statements to execute
     break;

     case default :
     //statements to execute
     break;
   }
   ```
   b.
   ```
   var myVariable = "test string";

   switch (myVariable) {
     case (first string) ;
     //statements to execute
     break:

     case (second string) ;
     //statements to execute
     break:

     case (third string) ;
     //statements to execute
     break:

     case (default) ;
     //statements to execute
     break:
   }
   ```

c.
```
var myVariable = "test string";

switch (myVariable) {
  case "first string" ;
  //statements to execute
  break:

  case "second string" ;
  //statements to execute
  break:

  case "third string" ;
  //statements to execute
  break:

  default ;
  //statements to execute
  break:
}
```

d.
```
var myVariable = "test string";

switch (myVariable) {
  case "first string" :
  //statements to execute
  break;

  case "second string" :
  //statements to execute
  break;

  case "third string" :
  //statements to execute
  break;

  default :
  //statements to execute
  break;
}
```

3. Describe the functionality of the JavaScript `continue` statement.

The `continue` statement is used to alter the way a loop executes. When a `continue` statement is encountered in a `for` loop or a `while` loop, the remaining statements in the code block are not executed. The flow of execution returns to the top of the loop where the loop's test condition is evaluated. If the test condition returns true, the code block executes. If the test condition returns false, flow of execution jumps to the statement following the closing curly brace of the loop.

INSTRUCTOR NOTE: In programming, statements that make decisions based on Boolean values and statements that cause code to execute repeatedly, or loop, are referred to as control structures. For control structures that use Boolean values, often a comparison operator is used in the pertinent expression. Encourage students to become completely familiar with JavaScript comparison operators, as they are at the core of JavaScript control structures.

Controlling Decisional Program Flow

In earlier lessons, you learned enough of the JavaScript language to declare variables, make assignments, and perform various types of arithmetic and string operations. Soon, you will learn more about working with the JavaScript document object model. However, until you learn the various techniques available for controlling decisional program flow, you will be unable to write any high-level JavaScript code. In this lesson, you will learn several methods of controlling the execution of JavaScript statements.

The *if...else* Statement

One of the most common needs in programming is the ability to branch to one of two processes, depending on the result of some test condition that you have scripted. For example, in a quiz, if the user answers a question correctly, one outcome may result. If the user answers incorrectly, a different outcome will result. The `if... else` statement allows you to script for such processes.

A single condition

In a simple `if` statement for which no other test condition exists, the syntax is as follows:

```
if (expression) {
  //statements to execute if expression is true
}
```

INSTRUCTOR NOTE: This section demonstrates to the students how to create `if` statements.

Note that the expression (the test condition) is enclosed in parentheses. A set of curly braces enclose a block of a single or multiple statements. These statements will execute only if the expression returns the Boolean value of true.

Following is a code snippet using an `if` statement:

```
if (userAnswer == 25) {
  alert("Correct answer.");
}
```

INSTRUCTOR NOTE: A common mistake for new JavaScript programmers is to use a single equal sign (=) instead of the correct comparison operator (==) when creating a Boolean expression for an `if` statement. Remind student to use the correct comparison operator in their Boolean expressions.

In this example, the user will see a message if the variable `userAnswer` equals 25.

Note that if the code associated with a test condition consists of just one line, you can omit the curly braces, as shown in the following examples:

```
if (userAnswer1 == 25) alert("Correct answer.");

if (userAnswer2 == "Yes")
  alert("Correct Answer");
```

This syntax is correct only if one line of code is to execute if the test condition is true. Note that a common syntax error is to forget the curly braces when designating a block of code for execution with an `if` statement. For this reason, always use the curly braces.

Multiple conditions

INSTRUCTOR NOTE: This section demonstrates to the students how to create an `else` clause in an `if` statement.

Suppose the user in the preceding example answers incorrectly. In this case, you will need a way to handle the possibility of a false value being returned by the test condition. Following is the syntax for handling multiple conditions:

```
if (expression) {
  //statements to execute if expression is true
}
else {
  //statements to execute if expression is false
}
```

INSTRUCTOR NOTE: Stress to students that in a simple `if` statement, nothing will happen if the Boolean expression returns a false value. Remind students that if code needs to execute when a false value is returned, it needs to be placed in an `else` clause.

Note that the `else` keyword is placed after the closing curly brace for the initial block of code associated with the test condition. The `else` block of code is enclosed in its own set of curly braces and will execute only if the test condition returns the Boolean value of false.

For example, suppose the correct answer is 25, but the user enters 30. Examine the following:

```
if (userAnswer == 25) {
  alert("Correct answer.");
}
else {
  alert("Your answer was incorrect. Try again!");
}
```

In this example, users will receive a message whether they answer correctly or incorrectly. Most situations will call for your code to respond to the test condition whether it is true or false, as shown in the previous example.

Using *if* for conditional program flow

The following example demonstrates the use of the `if...else` statement to decide whether to apply sales tax on purchases made in various U.S. jurisdictions. You will see that the code adds sales tax only to orders sent to California (CA) residents. (This example uses U.S. postal code abbreviations for state names.)

```
<HTML>
<HEAD>
<TITLE>Certified Internet Webmaster JavaScript Professional
</TITLE>

<SCRIPT>
<!--

// Define variables
var orderAmount = 50;
var state = "";

// Find out the user's shipping state

state = prompt("To which state will your order be shipped?","");

/*
If the user is in California, he or she will have to pay the total orderAmount
plus the additional 8.25% state sales tax. If not, they only have to pay the
actual orderAmount.
*/

if (state.toUpperCase() == "CA") {
  alert("Your total is: " + (orderAmount * 1.0825));
}
else {
  alert("Your total is: " + orderAmount);
}

//-->
</SCRIPT>

</HEAD>

<BODY>
<H1>Thank you for your order!</H1>

</BODY>
</HTML>
```

Additional if statements

If you want to test for multiple conditions, you can use additional `else if` statements. The following example demonstrates how to code such additional statements:

INSTRUCTOR NOTE: A common mistake that new JavaScript programmers make is to omit or confuse the opening or closing curly brace when using multiple `else if` clauses or embedded `if` statements. Stress the importance of carefully thinking through what each clause is intended to do as well as the importance of keeping track of opening and closing curly braces.

```
if (condition a) {
  //statements to execute
}
else if (condition b) {
  //statements to execute
}
else if (condition c) {
  //statements to execute
}
else {
  //statements to execute
}
```

Such multiple statements are quite useful in certain situations. For instance, the U.S. sales tax example could have included an additional `else if` statement to test for another state such as Hawaii (HI):

INSTRUCTOR NOTE: This section demonstrates to the students how to create else `if` clauses in `if` statement.

```
if (state.toUpperCase() == "CA") {
  alert("Your total is: " + (orderAmount * 1.0825));
}
   else if(state.toUpperCase() == "HI") {
     alert("Your total is: " + (orderAmount * 1.04));
}
else {
     alert("Your total is: " + orderAmount);
}
```

A programmer could add 49 `else if` blocks, one for each additional state. However, the `switch` statement, introduced by the JavaScript 1.2 standard, provides a more efficient way of coding in such situations. You will be introduced to the `switch` statement later in this lesson.

Multiple conditions in the same expression

So far, you have used only one expression in the test condition for the `if` statement. You can check multiple conditions in the same expression to attain a true or false condition. In the previous example, an `else if` block of code checked for another state that happened to have a different tax rate. Suppose both states had the same tax rate. The following example demonstrates the use of multiple conditions in the same expression to attain a true or false condition.

```
if ((state.toUpperCase() == "CA") || (state.toUpperCase() == ➘
"HI")) {
  alert("Your total is: " + (orderAmount * 1.0825));
}
else {
     alert("Your total is: " + orderAmount);
}
```

Note the use of the logical OR operator (||). In the example, if the `state` variable matches either CA (for California) or HI (for Hawaii), the expression will return a true value, and the appropriate tax is added. Any of the logical operators can be used in a multiple-condition expression.

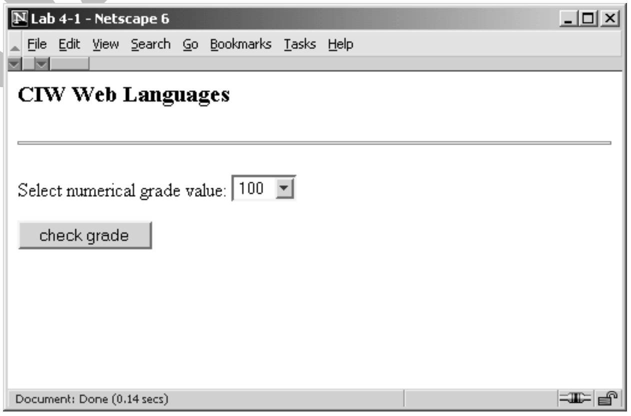
You will have an opportunity to use `if` statements in the next lab.

Lab 4-1: Using *if* statements

INSTRUCTOR NOTE: The code for this lab demonstrates how to extract a value from a `select` object using JavaScript. This concept will be discussed elsewhere in the book.

In this lab, you will create an `if` statement. The `if` statement will test an integer for a particular range of values. An alert dialog box will provide pertinent information if the integer falls within this range.

1. **Editor:** Open **lab4-1.htm** from the Lesson 4 folder of the Student_Files directory.
2. **Editor:** A function named `checkgrade()` has been started for you. A variable named `myGrade` is assigned an integer value received from a `select` object in an HTML form.
3. **Editor:** Following the `myGrade` initialization statement, create an `if` statement. As the condition for the `if` statement, test the `myGrade` variable for a value greater than or equal to 91. If the value is greater than or equal to 91, output an alert that informs the user that his or her grade is an A.
4. **Editor:** Save **lab4-1.htm**.
5. **Browser:** Open **lab4-1.htm**. Your screen should resemble Figure 4-1.



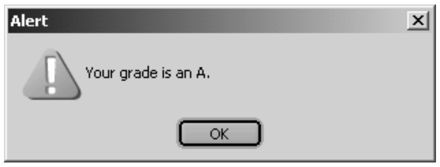
CIW Web Languages

Select numerical grade value: 100

Document: Done (0.14 secs)

Figure 4-1: Lab4-1.htm

6. **Browser:** Select a numerical grade value of 91 or greater from the drop-down menu. You should see an alert resembling Figure 4-2.


Alert

Your grade is an A.

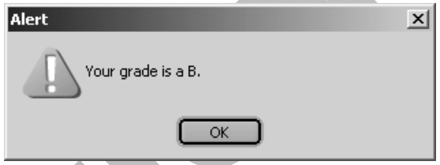
Figure 4-2: Alert dialog box

7. **Browser:** Click **OK** to close the alert.

8. **Editor:** Add an `else if` clause to the `if` statement. As the condition for the `else if` clause, test the `myGrade` variable for a value greater than or equal to 81 and less than or equal to 90. Hint: Use the logical AND operator (&&) to perform this test. If the `myGrade` variable falls within this range, output an alert dialog box informing the user that his or her grade is a B.

9. **Editor:** Save **lab4-1.htm**.

10. **Browser:** Refresh **lab4-1.htm**. Select a numerical grade value between 81 and 90 from the drop-down menu. You should see an alert resembling Figure 4-3.


Alert

Your grade is a B.

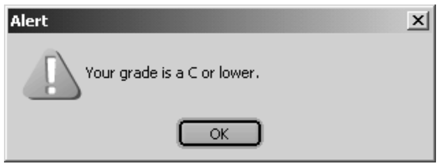
Figure 4-3: Alert dialog box

11. **Browser:** Click **OK** to close the alert.

12. **Editor:** You could continue adding `else if` clauses to test, then map all possible ranges of integer values to a letter grade. However, for now, add an `else` clause to the `if` statement. The `else` clause will simply output an alert dialog box informing the user that his or her grade is a C or lower.

13. **Editor:** Save **lab4-1.htm**.

14. **Browser:** Refresh **lab4-1.htm**. Select a numerical grade value of 80 from the drop-down menu. You should see an alert resembling Figure 4-4.


Alert

Your grade is a C or lower.

Figure 4-4: Alert dialog box

15. **Browser:** Click **OK** to close the alert.

This lab provided an opportunity to create and use an `if...else if...else` statement. You tested an integer's value and displayed an appropriate message when the value fell into a particular range.

The *while* Statement

The `while` statement is used to execute a block of code for as long as (while) a certain test condition is true. The syntax is as follows:

```
while (expression) {
//one or more statements to execute
}
```

INSTRUCTOR NOTE: This section demonstrates to the students how to create `while` statements. Also, the `isNan()` function is demonstrated in this section.

When program execution reaches the `while` statement, the test condition is evaluated. If the expression returns the Boolean value of true, the statements encompassed within the curly braces are executed. Then, program flow returns to the beginning of the `while` statement, where the test condition is re-evaluated. The code block will repeat itself, or loop, for as long as the expression returns the value of true. The `while` loop is especially useful when you are not sure how many times the loop will need to execute.

An improperly coded `while` statement can set up an infinite loop that could cause the browser to wait indefinitely. Therefore, be sure to include some statement within the `while` condition to ensure that at some point the test condition will become false, thus terminating the loop.

For example, the following loop continues until the value of the variable `num` is 10:

```
//declare and initialize variable
var num = 0;

//start while loop and create test condition
while (num < 10) {

  //output value of variable
  document.write(num + " ");

  //increment variable
  num++;

} //end while loop
```

INSTRUCTOR NOTE: Remind students that if the test condition of a `while` loop depends on the value of a loop counter, it is up to the programmer to manually declare and initialize the loop counter variable before the loop as well as increment or decrement the loop counter inside the loop.

In this example, if you forgot to include the increment statement (`num++`), an infinite loop would result. The variable `num` would remain at zero, and thus would always be less than 10. The `while` loop would continue until the browser notified the user of an unsafe condition, or the browser may wait indefinitely, forcing the user to quit the browser.

Just as with `if` statements, it is possible to have multiple test conditions in the `while` expression. Consider the following example:

```
while ((condition1) || (condition2)) {
   //statements to execute
}
```

As long as one condition or the other returns true, the loop will continue.

A `while` loop can be used in many different situations. In the following lab, the user will enter a numerical grade value in a text box. You will use a `while` loop to test the user's input. The lab will introduce the `isNaN()` function. The `isNaN()` function is used to determine whether a given value is a valid number. The following example demonstrates the `isNaN()` function:

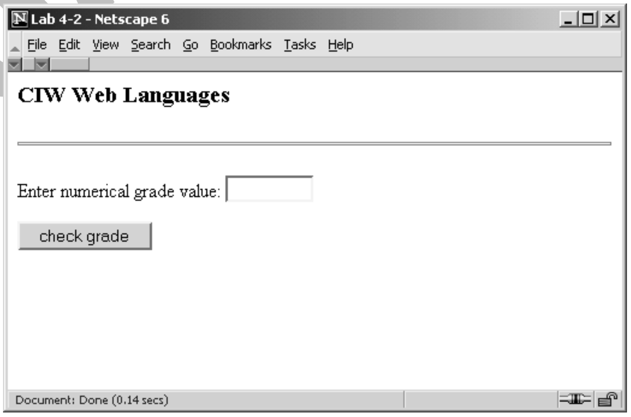
```
var myString = "A string of text.";
var myNumber = 10;

alert(isNaN(myString)); //displays true

alert(isNaN(myNumber)); //displays false
```

You can think of the isNaN() function as "is not a number." In the preceding example, a true value is returned for myString because it is not a number. A false value is returned for myNumber because it is a numeric value.

Lab 4-2: Using a *while* statement

INSTRUCTOR NOTE: The code for this lab demonstrates how to extract a value from a textbox object using JavaScript. This concept will be discussed in depth elsewhere in the book.

1. **Editor:** Open **lab4-2.htm** from the Lesson 4 folder of the Student_Files directory.
2. **Editor:** Locate the checkGrade() function in the <HEAD> section of the document. This function contains essentially the same code as lab4-1.htm. However, the user enters a numerical grade value in a text box instead of selecting one from a drop-down menu.
3. **Editor:** Create a while loop after the myGrade initialization statement. In the condition for the loop, use the isNaN() function to test the myGrade variable. In pseudo-code, the first line of the while statement should read: *while is not a number (myGrade).*
4. **Editor:** One line of code is needed inside the loop. If myGrade is not a number, reassign myGrade the return value of a prompt dialog box that asks the user to input a numerical value. Do not forget to use the parseInt() function to convert the string value returned by the prompt to an integer. The while loop will continue until the user enters a number.
5. **Editor:** Save **lab4-2.htm**.
6. **Browser:** Open **lab4-2.htm**. Your screen should resemble Figure 4-5.



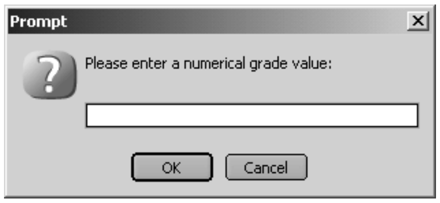
CIW Web Languages

Enter numerical grade value:

Document: Done (0.14 secs)

Figure 4-5: Lab4-2.htm

7. **Browser:** Entering a numerical value between 0 and 100 in the text box and clicking the check grade button will result in the same output as lab4-1.htm. Enter a nonnumerical value in the text box and click the **check grade** button. You should see a prompt similar to Figure 4-6.



Please enter a numerical grade value:

Figure 4-6: Prompt dialog box

8. **Browser:** Click **OK** or **Cancel** without entering any information, or enter a nonnumerical value. You should see the prompt again. In fact, the prompt dialog box should repeatedly appear until a numerical value is entered. After a numerical value is entered, you should see the same output as produced by lab4-1.htm.

   This lab is intended to demonstrate a `while` loop, and you have seen the effect of using a loop to repeatedly ask for user input. However, good JavaScript programming practices suggest that you do not prevent a user from canceling a prompt dialog box as shown in lab4-2.htm. A well-designed program would ask for user input a predetermined number of times. If the user repeatedly clicks Cancel, at some point the program should exit. The next portion of this lab will demonstrate this concept.

9. **Editor:** Open **lab4-2.1.htm**. This is the same program as lab4-2.htm with a minor improvement. A variable named `attempts` has been declared and initialized with a value of 1.

10. **Editor:** In the conditional expression for the `while` loop, also test the `attempts` variable for a value less than or equal to 2. Use the logical AND operator to perform this task.

11. **Editor:** Add a statement inside the loop that increments the `attempts` variable each time through the loop.

12. **Editor:** Note that an additional statement has been provided for you after the `while` loop. An `if` statement tests the `myGrade` and `attempts` variables. If `myGrade` is still not a number, and the `attempts` variable equals 3, a `return` statement is used to exit the function.

13. **Editor:** Save **lab4-2.1.htm**.

14. **Browser:** Open **lab4-2.1.htm**. Experiment with the page. If the user does not enter the required data in two attempts, the prompt dialog boxes should disappear with no further output from the program.

As you can imagine, misuse of the `while` loop can cause frustration and create problems. It is important to remember that the value of the condition under which the `while` statement executes must become false at some point. Otherwise, the `while` statement will loop indefinitely, and the user will have to exit the browser program or reboot to escape the loop.

WARNING! *Note that lab4-2.htm was for demonstration purposes only. It is not good programming practice to prevent the user from selecting the Cancel button on a prompt dialog box. This type of programming could cause users to avoid using your Web application. However, you should always program for the possibility that the user may select the Cancel button.*

The *do...while* Statement

INSTRUCTOR NOTE: The `do...while` statement is available only with Netscape Navigator 4.0 and later and with Microsoft Internet Explorer 4.0 and later.

The `do...while` statement was introduced with the JavaScript 1.2 specification. It operates like the `while` statement, with one key difference: The `do...while` statement does not check the conditional expression until after the first time through the loop, guaranteeing that the code within the curly braces will execute at least once.

The syntax for the `do...while` statement is as follows:

```
do {
//statements to execute
} while (conditional expression)
```

The following example demonstrates the `do...while` statement:

```
//Declare and initialize variable
var num = 10;

//Start do...while loop
do{
     //Output value of variable
    document.write(num + " ");
     //Increment variable
    num++;
} while (num < 10) //End do...while loop
                   //when condition is false
```

Note that in this example, the variable `num` is initialized with the value of 10. The `do...while` loop is scripted to execute as long as `num < 10`. Even though the `num` variable is never less than 10, the loop will execute one time, outputting the value of `num` to the screen. The `do...while` loop would then end with control of the program passing to the statement after the `while` expression.

Note also that a `do...while` loop can accomplish some tasks that a `while` loop cannot. Remember that the difference between the two control structures is that the `do...while` loop will execute its code at least one time, regardless of the Boolean value returned by the test condition.

The *for* Statement

The primary purpose of the `for` statement is to repeat a group of statements for some particular range of values. The format of a `for` statement is as follows:

```
for (x; y; z) {
  //one or more statements
}
```

In this example, x is the loop counter variable initialization expression, y is the condition under which the loop will repeat, and z is the expression that increments or decrements the loop counter. Consider the following for statement:

```
for (var i = 0; i < 5; i++) {
  document.write("Loop number: " + i + "<P>");
}
alert("The final value if i is " + i);
```

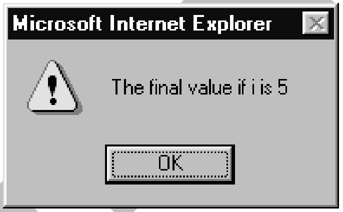
INSTRUCTOR NOTE: This section demonstrates to the students how to create for statements.

In this example, a variable named i is declared and initialized as the loop counter variable. Note that syntax requirements allow you to use the var keyword to declare the loop counter variable inside the parentheses, although JavaScript does not require you to do so.

Next, the condition under which the loop will repeat is declared. In this case, as long as i < 5, the loop will continue.

The next expression, i++, increments the loop counter variable by the value of 1 after each execution of the loop. When i reaches the value of 5, it will no longer be less than 5, which is the test condition in the for statement. Thus the loop will end, passing control to the next line: the alert statement.

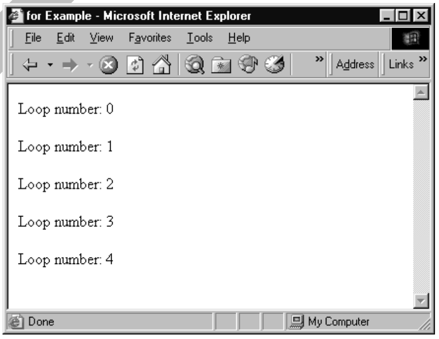
When this code is executed, the user will see a message similar to Figure 4-7.



```
The final value if i is 5
```

Figure 4-7: for loop alert message

The user would also see a browser screen similar to Figure 4-8.


for Example - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
Done
My Computer

```
Loop number: 0

Loop number: 1

Loop number: 2

Loop number: 3

Loop number: 4
```

Figure 4-8: Output of for loop

Note that the loop counter variable was initialized to the numeric value 0. The first time through the loop, 0 was output to the screen, illustrating the fact that the first time the for statement is executed, the increment expression is ignored. The loop counter variable is not incremented until after the code inside the for statement has executed one time.

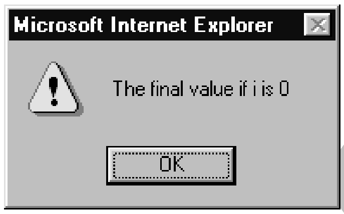
Tech Note *If you do not want your loop counter variable to start at 0, simply initialize the loop counter variable to the numeric value of 1, or any other value that you may need.*

Note that, if needed, the loop counter mechanism can be executed in a backward fashion. Consider the following example.

```
for (var i = 5; i > 0; i--) {
  document.write("Loop number: " + i + "<P>");
}
alert("The final value if i is " + i);
```

In this example, the loop counter variable `i` is initialized with a value of 5. The loop will continue as long as `i` is greater than zero. The loop counter variable is decremented after each execution of the loop.

When this code is executed, the user will see a message similar to Figure 4-9.

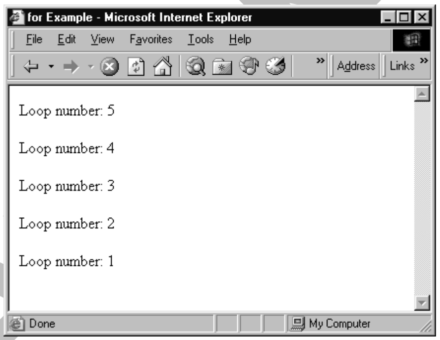


```
The final value if i is 0
```

Figure 4-9: `for` loop alert message

INSTRUCTOR NOTE: Refer to **Instructor Note 4-1** for more information concerning JavaScript `for` statements.

The user would also see a browser screen similar to Figure 4-10.


for Example - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
Done
My Computer

```
Loop number: 5

Loop number: 4

Loop number: 3

Loop number: 2

Loop number: 1
```

Figure 4-10: Output of `for` loop

The next lab demonstrates the use of a `for` loop. You will use the `for` statement to output HTML and the value of the loop counter variable.

Lab 4-3: Using a *for* statement

In this lab, you will use a `for` statement to output HTML and the value of the loop counter variable. You will use a `for` loop to create a drop-down menu that contains the values 100 decreasing to zero.

1. **Editor:** Open **lab4-3.htm** from the Lesson 4 folder of the Student_Files directory.

2. **Editor:** This file is essentially the same as lab4-1.htm. Locate the second `<SCRIPT>` block in the `<BODY>` section of the document. Create a `for` statement. Create a variable named `i` for the loop counter variable. Assign an initial value of 100 to `i`. The loop will continue as long as `i` is greater than or equal to zero. Use the decrement operator (`--`) to decrement `i` by one each time through the loop.

3. **Editor:** Inside the loop, use a `document.write()` statement to output an `<OPTION>` tag. Concatenate the value of `i` into the expression. The value of the loop counter variable will supply the text for the drop-down menu.

4. **Editor:** Save **lab4-3.htm**.

5. **Browser:** Open **lab4-3.htm**. Your screen should resemble Figure 4-1. Click on the drop-down menu to open it, and scroll through the items in the drop down menu. Your screen should resemble Figure 4-11.

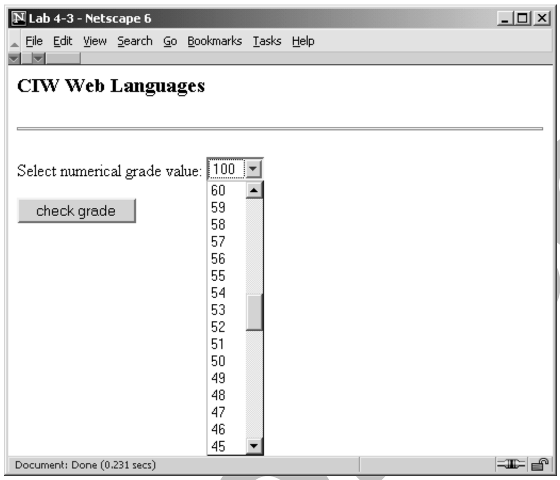

Lab 4-3 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
CIW Web Languages
Select numerical grade value:
100
60
59
58
57
56
55
54
53
52
51
50
49
48
47
46
45
check grade
Document: Done (0.231 secs)

Figure 4-11: Lab4-3.htm

6. **Browser:** The program should respond in a manner similar to lab4-1.htm when a numerical grade value is selected from the drop-down menu.

This lab demonstrated the ease with which a few lines of JavaScript code generated 100 lines of HTML code. The `for` statement can be used to automate many tasks in many different situations.

The *break* Statement

INSTRUCTOR NOTE: This section demonstrates to the students how to create and use a `break` statement. Refer to **Instructor Note 4-2** for more information concerning the `break` statement.

A way to exit a loop that would otherwise continue to execute is to use the `break` statement. Usually, you will find the `break` keyword inside an `if` clause. If a certain condition is reached, the program will break out of the loop. If not, the loop will continue. You can use the `break` statement within the `for`, `while` and `do...while` loops, as well as with the `switch` statement.

The following example demonstrates the `break` statement:

```
for(var i = 0; i <= 10; i++){
  document.write("Loop number: " + i);
  if(i == 4) break;
}
```

In this example, the `for` loop will terminate when the loop counter variable `i` equals 4, even though the test condition reads `i <= 10`.

nested statement
A common programming practice of embedding a script block within another script block.

In the next lab, you will create code that includes a `while` statement and `if` **nested statements**. The program will display a prompt dialog box that asks for input. If the user enters any value, the loop will end and the program will reflect the input to the user. You will add code to test for a `null` value, in which case you will invoke a `break` statement to end the loop.

Lab 4-4: Nesting *if* and *break* statements inside a *while* loop

INSTRUCTOR NOTE: See **Instructor Note 4-3** for information concerning labels and the `break` statement.

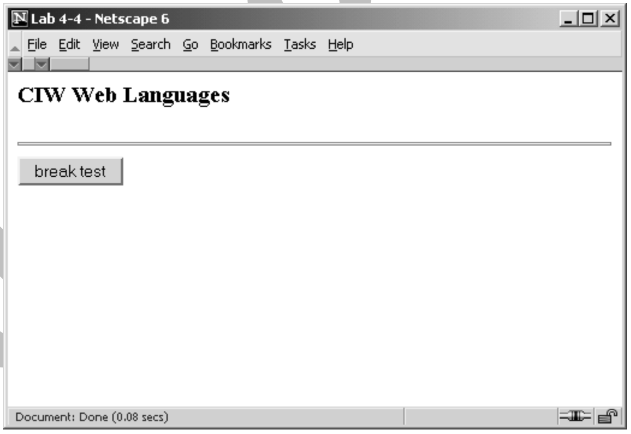
1. **Editor:** Open the **lab4-4.htm** file from the Lesson 4 folder of the Student_Files directory.

2. **Editor:** Locate the <SCRIPT> block in the <HEAD> section of the document. A function named `breakTest()` has been started for you. Two variables have been declared: `loopBoolean` and `myValue`. The `loopBoolean` variable is set to true and is used to determine when the `while` loop will end. The `myValue` variable is assigned an empty string in anticipation of the user's input.

3. **Editor:** After the variable declarations, a `while` loop is started. After the `myValue` variable receives its value, add an `if` statement that tests `myValue` for a `null` value. If it is `null`, invoke a `break` statement.

4. **Editor:** Save **lab4-4.htm**.

5. **Browser:** Open **lab4-4.htm**. Your screen should resemble Figure 4-12.


Lab 4-4 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
break test
Document: Done (0.08 secs)

CIW Web Languages

Figure 4-12: Lab4-4.htm

6. **Browser:** Click the **break test** button. You should see a prompt dialog box as shown in Figure 4-13.

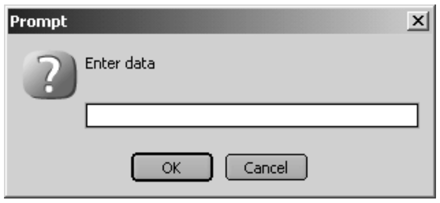


Figure 4-13: Prompt dialog box

7. **Browser:** Click **OK** without entering any data. The prompt dialog box should reappear whenever this action is taken. Click **Cancel**. The program should recognize the `null` value returned when this action is taken and the `break` statement should end the program.

8. **Browser:** Any data entered in the prompt dialog box should appear in an alert dialog box.

The *continue* Statement

The `continue` statement is used to force the flow of control back to the top of a loop. You can think of `continue` as a "skip" or bypass command. When execution hits a `continue` statement, all statements between it and the end of the loop block are skipped, and execution returns to the top of the loop. The test condition for the loop is then evaluated to determine whether the loop should execute again. The `continue` statement can be used only within a `for` or a `while` loop. The next section provides an example of a `continue` statement used in a `while` loop.

Using *continue* in a *while* loop

INSTRUCTOR NOTE: This section demonstrates to the students how to create and use a `continue` statement.

In the following example, the `continue` statement is used to manipulate a `while` loop.

```
<HTML>
<HEAD>
<TITLE>Certified Internet Webmaster JavaScript Professional</TITLE>

<SCRIPT LANGUAGE="JavaScript">
<!--

  var leapYears = 0;
  var curYear = 2001;

  while (curYear <= 2100) {
    curYear++;

if (curYear % 4) continue;

    leapYears++;
    document.write(curYear + " ");

  }

// -->
</SCRIPT>
</HEAD>
<BODY>
<H3>The number of leap years between 2001 and 2100 is:
<SCRIPT LANGUAGE="JavaScript">
<!--

  document.write(leapYears);

// -->
</SCRIPT>
</H3>
</BODY>
</HTML>
```

In this example, the script loops through each year from 2001 to 2100. The `curYear` variable holds the current year. The `leapYears` variable is used to count the number of leap years.

In the program, a `while` loop is started and will continue as long as the `curYear` variable is less than or equal to 2100. When the year is evenly divisible by 4, the `leapYears` variable is incremented by 1 and the value currently held by the `curYear` variable is output to the screen. In the years that are not evenly divisible by 4, the increment of the `leapYears` variable and the `document.write()` statements are skipped because of the `continue` statement. In the end, the `leapYears` variable contains the count of all years from 2001 to 2100 that are evenly divisible by 4, which is the correct count for the number of leap years in that period. Note the use of the modulus operator: %. This operator divides the left operand by the right operand and returns the remainder. Note this portion of the code:

```
(curYear % 4)
```

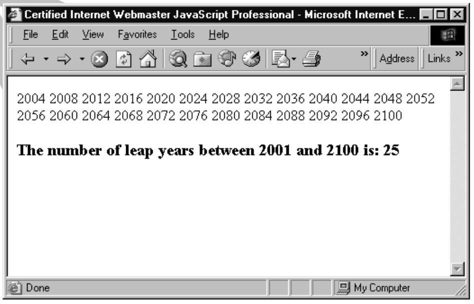
The use of the modulus operator will be easier to understand if an actual value is inserted for the `curYear` variable. The first time through the loop, this statement will appear as: `(2002 % 4)`. 2002 divided by 4 is 500 with a remainder of 2, which is then the return value from the statement. After evaluation of the statement, the `if` statement reads as follows:

```
if (2) continue;
```

Any non-zero integer will evaluate to true in JavaScript. Thus the statement is true and the `continue` statement is invoked, skipping the rest of the statements in the loop. The only time the `continue` statement will not be invoked is when zero is returned from the calculation. A zero will evaluate to false, and the rest of the code in the loop will execute.

Note that the formula shown in the previous example is not rigorous enough to count leap years for an unlimited time range, but is sufficient for the time period in this example.

If you were to load this source code into your browser, your screen would resemble Figure 4-14.


Certified Internet Webmaster JavaScript Professional - Microsoft Internet E...
File Edit View Favorites Tools Help
Address Links
Done
My Computer

```
2004 2008 2012 2016 2020 2024 2028 2032 2036 2040 2044 2048 2052
2056 2060 2064 2068 2072 2076 2080 2084 2088 2092 2096 2100

The number of leap years between 2001 and 2100 is: 25
```

Figure 4-14: Leap year example

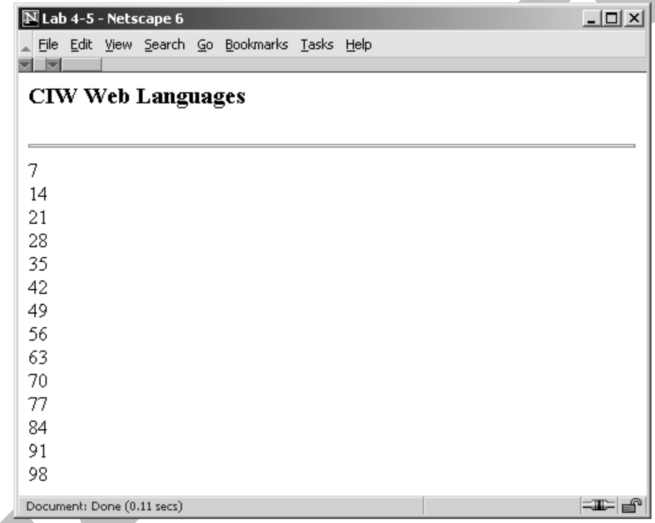
The following lab will provide an opportunity to use the `continue` statement. The small program in this lab will use the `continue` statement to output all numbers between 0 and 100 that are evenly divisible by 7.

INSTRUCTOR NOTE: The `continue` statement can be used with labels in a manner similar to that demonstrated with the `break` statement.

Lab 4-5: Using a *continue* statement

This lab will use the `continue` statement to manipulate a `for` loop.

1. **Editor:** Open **lab4-5.htm** from the Lesson 4 folder of the Student_Files directory.
2. **Editor:** A `for` loop has been defined that outputs the numbers 1 through 100. Before the `document.write()` statement, create an `if` statement that tests the loop counter variable `i`. If `i` divided by 7 returns a remainder, invoke a `continue` statement. The only numbers that should be output are those evenly divisible by 7. Hint: Use the modulus operator (%) to perform this task.
3. **Editor:** Save **lab4-5.htm**.
4. **Browser:** Open **lab4-5.htm**. Your screen should resemble Figure 4-15.


Lab 4-5 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
Document: Done (0.11 secs)

CIW Web Languages

```
7
14
21
28
35
42
49
56
63
70
77
84
91
98
```

Figure 4-15: Lab4-5.htm

The *switch* Statement

Earlier in this lesson, you were introduced to the `if` statement. You learned that you could include multiple `else if` clauses to evaluate various test conditions. If the first condition returned a false value, another condition was evaluated, and then another, and so on until a true value was found. If no true value was found, the `else` clause provided code that would execute in that situation. An example was shown earlier in this lesson that checked user input for a U.S. state to determine state sales tax. Suppose you needed code that checked for all 50 U.S. states. A more efficient method for evaluating a large number of test conditions is the `switch` statement.

INSTRUCTOR NOTE: This section demonstrates to the students how to create a `switch` statement. The students may apply this information in **Optional Lab 4-1: Using a *switch* statement**.

The `switch` statement was introduced with the JavaScript 1.2 specification. This statement compares a value against other values, searching for a match. If a match is found, the code associated with the match will execute. The `break` statement is then used to exit the `switch` block of code.

Essentially, `switch` functions the same as multiple `else if` clauses within an `if` statement. However, `switch` is more readable and allows you to specify a default set of statements to execute if no match is found. The `default` clause is similar in functionality to an `else` clause in an `if` statement.

The syntax for creating a `switch` statement is as follows:

```
switch (expression) {
  case "str1" :
  //statements to execute
  break;

  case "str2" :
  //statements to execute
  break;

  case "str3" :
  //statements to execute
  break;

  default :
  //statements to execute
  break;
}
```

The following example demonstrates the use of the `switch` statement. This example uses U.S. postal code abbreviations for state names.

```
<HTML>
<HEAD>
<TITLE>Certified Internet Webmaster JavaScript Professional</TITLE>

<SCRIPT LANGUAGE="JavaScript">
<!--

// Define variables
var orderAmount = 50;
var state = "";

// Find out the user's shipping state

state = prompt("To which state will your order be shipped?","");

//Add appropriate tax for state

switch (state.toUpperCase()) {

  case "CA" :
  alert("Your total is: " + (orderAmount * 1.0825));
  break;

  case "HI" :
  alert("Your total is: " + (orderAmount * 1.04));
  break;

  //Note that the same code executes for both cases
  case "WA" :
  case "NV" :
  alert("Your total is: " + (orderAmount * 1.065));
  break;
```

```
    case "AZ" :
    alert("Your total is: " + (orderAmount * 1.05));
    break;

    case "UT" :
    alert("Your total is: " + (orderAmount * 1.0475));
    break;

    default :
    alert("Your total is: " + orderAmount);
    break;
  }

  //-->
  </SCRIPT>
  </HEAD>

  <BODY>
  <H1>Thank you for your order!</H1>

  </BODY>
  </HTML>
```

In this example, the `switch` keyword is followed by the expression against which the `case` statements will be compared. Here, the expression is the `state` variable with the value supplied by the user and converted to uppercase via the `toUpperCase()` method.

INSTRUCTOR NOTE: Refer to **Instructor Note 4-4** for more information concerning the `switch` statement.

After the user enters a state, the `switch` statement searches the `case` statements. Note that a colon, not a semicolon, follows the `case` and `default` keywords. This syntax is required for the `switch` statement. When a match is found, the associated code will execute. If no match is found, the code associated with the `default` statement will execute.

To further explain the `switch` statement, consider what happens when the code executes; this section of the `switch` statement would appear as the following (if the user entered CA):

```
  ...
  // The text CA is supplied by the JavaScript interpreter
  // as the value for the state variable.
  switch (CA) {

    case "CA":
    alert("Your total is: " + (orderAmount * 1.0825));
    break;
  ...
  }
```

After execution of the associated code, the `break` keyword ends the `switch` statement, with control of the program passed to the line following the closing curly brace. The entire `switch` block of code is enclosed by one set of curly braces, an advantage of the `switch` statement over multiple `else if` clauses in an `if` statement. Only one set of curly braces is needed to enclose the entire code block.

As this example demonstrates, it is possible to have the same code execute for different `case` statements by listing those `case` statements together. In this example, the states Washington (WA) and Nevada (NV) share the same tax rate. Note that no `break` keyword follows `case "WA":`. This coding essentially allows execution of the code to "fall through" to the code underneath `case "NV":`.

The optional lab for this lesson demonstrates the use of the `switch` statement. Remember that this construct was introduced with JavaScript 1.2 and requires a version 4 (or later) browser to execute properly.

Lesson Summary

Application project

These projects will challenge you to use some of what you have learned in this lesson. Create an HTML page with a `<SCRIPT>` block in the `<HEAD>` section of the document. Create a function named `ifElseTest()`. Inside the function, create an `if...else` statement. Use the return value of a confirm dialog box as the test expression for the `if` statement. If the `confirm()` method returns true, create code that reflects this result to the user. Use the `else` clause to execute some code if the `confirm()` method returns false. Call the `ifElseTest()` function using the `onLoad` event handler inside the `<BODY>` tag.

Create an HTML page with a `<SCRIPT>` block in the `<HEAD>` section of the document. Create a function named `whileTest()`. Inside the function, create a variable named `number` and assign it a value between one and 10. Create another variable named `answer` and assign it a value of 0 (zero). Then create a `while` loop. Create code that will cause the loop to execute as long as the `number` variable does not equal the `answer` variable. Inside the loop, assign the `answer` variable the return value from a prompt dialog box. The prompt will ask the user to guess a number between one and 10. The loop will continue until the proper answer is entered. After the loop exits, use an alert dialog box to inform the user of a correct guess. After you have the code working properly, create code that will allow the user only three guesses. If, after three guesses, the user has not entered the proper information, exit the function and alert the user that he or she is out of guesses via an alert dialog box. Ensure that only one dialog box appears after the function is exited, one with a correct guess message, or one asking the user to try again. Experiment with different methods that you have seen for calling the function. You can use the `load` event or the `onClick` event handler of a form button.

Next, create an HTML page with a `<SCRIPT>` block in the `<HEAD>` section of the document. Create a `for` loop that will output the numbers one through 200. Create code that ensures there is a space between each number. Create code that will cause a line break after the output reaches 10, 20, 30, 40 and so on. Create code that outputs the value of the loop counter variable after the `for` loop is exited.

Create an HTML page with a `<SCRIPT>` block in the `<HEAD>` section of the document. Create a variable named `color`. Assign the result of a prompt dialog box as the value for the `color` variable. The prompt dialog box will ask the user to select red, white or blue. Create a `switch` statement. The test expression is the `color` variable converted to uppercase. Use RED, WHITE and BLUE after the `case` keywords. As the code executes, reflect back to the user the selected choice. Add a default clause that will execute if the user enters a choice other than those offered. Remember that the user may click Cancel on the prompt dialog box. Create code before the `switch` statement to handle this eventuality. Experiment with different ways of handling all the possible actions the user might take.

Skills review

The `if... else` statement allows you to branch to one of two processes depending on the result of some test condition. If you want to test for multiple conditions, you can use additional `else if` statements. The `while` statement is used to execute a code group for as long as a certain test condition is true. The `do...while` statement operates exactly like the `while` statement, with one key difference: The `do...while` statement does not check the conditional expression until after the first time through the loop. The `for` statement is used to repeat a group of statements for an explicit range of values. The `break` statement is used to exit a loop that would otherwise continue to execute. The `continue` statement is used to force the flow of control back to the top of a loop. The `switch` statement compares a value against other values, searching for a match. If a match is found, the code associated with the match will execute.

Now that you have completed this lesson, you should be able to:

- ✓ Use the `if...` statement.
- ✓ Use the `while...` statement.
- ✓ Use the `for...` statement.
- ✓ Use the `break` and `continue` statements.
- ✓ Define the `do...while` statement.
- ✓ Use the `switch...` statement.

Lesson 4 Review

1. What is the purpose of the `if...else` statement?

   The `if...else` statement allows you to branch to one of two processes depending on the result of some test condition that you have scripted.

2. What syntax error is common when writing `if` statements?

   Forgetting the curly braces when designating a block of code for execution.

3. Where should the `else` keyword be placed in an `if...else` statement?

   The `else` keyword is placed after the closing curly brace for the `if` block of code associated with the test condition. The `else` block of code is enclosed in its own set of curly braces.

4. What is the purpose of the `else if` statement?

   You can use additional `else if` statements to test for multiple conditions in an `if` statement.

5. What is the purpose of the `while` statement?

   The `while` statement is used to execute a block of code for as long as a certain test condition is true.

6. What can happen if a `while` statement is improperly coded? How can you avoid this mistake?

   An improperly coded `while` statement can set up an infinite loop that could cause the browser to wait indefinitely. To avoid this mistake, include some statement within the `while` condition to ensure that at some point the test condition will become false, thus terminating the loop.

7. What is the purpose of the `for` statement?

   The `for` statement is used to repeat a group of statements for some particular range of values.

8. What is the `isNaN()` method?

   The `isNaN()` method ("is not a number") determines whether a given value is a valid number.

9. What is nesting?

   Nesting is a common programming practice of embedding a script block within another script block.

10. What is the purpose of the `break` statement?

    The `break` statement is used to exit a loop that would otherwise continue to execute.

11. What is the purpose of the `continue` statement?

    The `continue` statement is used to force the flow of control back to the top of a loop. It acts like a "skip" or bypass command.

12. What is the purpose of the `switch` statement? To what type of statement is it similar?

    The `switch` statement is used to compare a value against other values, searching for a match. If a match is found, the code associated with the match is found. The `switch` statement functions the same as multiple `else if` clauses within an `if` statement.

13. What is the purpose of the `do...while` statement? How does it differ from the `while` statement?

    Like the `while` statement, the `do...while` statement is used to execute a block of code for as long as a certain test condition is true. However, the `do...while` statement does not check the conditional expression until after the first time through the loop, guaranteeing that the code within the curly braces will execute at least once.

Lesson 4 Instructor Section

This section is a supplement containing additional tasks for students to complete in conjunction with the lesson. It also contains additional instructor notes. The instructor may use all, some or none of these additional tools, as appropriate to the specific learning environment. These elements are:

- **Additional Instructor Notes**
  Detailed instructor notes that expand the tips and suggestions presented in the instructor margin notes throughout the lesson.
- **Optional Labs**
  Computer-based labs to be completed during class or as homework.
- **Lesson Quiz**
  Multiple-choice test to assess student knowledge of lesson material.

Additional Instructor Notes

The following section contains detailed instructor notes that expand the tips and suggestions presented in the instructor margin notes throughout the lesson.

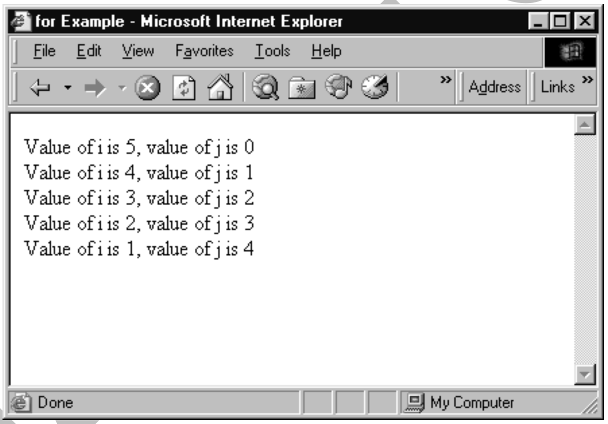
Instructor Note 4-1

Location: The for *Statement*

If needed, a `for` statement can use more than one loop counter variable. Consider the following example:

```
var i, j;
for (i = 5, j = 0; i > 0, j < 5; i--, j++) {
  document.write("Value of i is " + i + ", value of j is " ➘
                 + j + "<BR>");
}
```

As many loop counter variables as needed can be used. Note that the loop counter variables are in a comma-delimited list in each section of the `for` statement's definition. The output of this example is shown in Figure IN4-1.


for Example - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
Done
My Computer

```
Value of i is 5, value of j is 0
Value of i is 4, value of j is 1
Value of i is 3, value of j is 2
Value of i is 2, value of j is 3
Value of i is 1, value of j is 4
```

Figure IN4-1: Result of `for` *example*

Instructor Note 4-2

Location: The break *Statement*

A common use of the `break` statement is to end a loop that is used to search for a particular value. When that value is found, the appropriate code executes and the loop is exited. Consider the following example:

```
<HTML>
<HEAD>
<TITLE>break Example</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
  var shows = new Array("The Beverly Hillbillies", ➘
  "The Munsters", "Leave It To Beaver", "The Honeymooners", ➘
  "I Love Lucy");

  var mainCharacters = new Array("Jed Clampett", ➘
  "Herman Munster", "Beaver Cleaver", "Ralph Kramden", ➘
  "Lucy Ricardo");
  var count = shows.length;

  function findMainCharacter(form) {
    var lookFor = form.input.value;
```

```
      for(var i = 0; i < count; i++) {
        if(lookFor.toUpperCase() == shows[i].toUpperCase()) {
          break;
        }
      }

      if(i < shows.length) {
        alert("The main character is " + mainCharacters[i]);
      }
      else {
        alert("The main character was not found for " + lookFor);
      }
    }

//-->
</SCRIPT>

</HEAD>

<BODY>

<FORM NAME="showSearch">
Enter a show's name:
<INPUT TYPE="text" NAME="input"><BR>
<INPUT TYPE="button" VALUE="Search" onClick="findMainCharacter(this.form);">
</FORM>

</BODY>
</HTML>
```

Note this portion of the code:

```
for(var i = 0; i < count; i++) {
  if(lookFor.toUpperCase() == shows[i].toUpperCase()) {
    break;
  }
}
```

If the search text is found, the loop is broken. Code then checks to ensure that the loop counter variable did not go beyond the available number of array values. If the loop counter does not go beyond the end of the array, the user is presented with the appropriate information. If the loop ended because no match was found, the user is informed that no data was found.

This example uses concepts that are introduced elsewhere in this course. It may be beneficial to use this example after form handling and arrays have been introduced. Also, note that the functionality of this example is used in the application discussed in the Custom JavaScript Objects lesson. This example is included in this lesson because the example demonstrates a very common use of the `break` keyword inside a loop construct.

Instructor Note 4-3

Location: The break *Statement*

Students who are experienced in other programming languages may ask about labels and `break` statements. Labeled statements are supported in JavaScript. The following is an example.

```
var outerBoolean;
var innerBoolean;

outer :
for (var i = 1; i <= 3; i++) {
  outerBoolean = ➘
  confirm("The loop counter variable i = " + i + ".\n" +
          "The loop will continue as long as i <= 3.\n" +
          "Click cancel to invoke a break statement.","")

  if (outerBoolean == false) {
    innerBoolean = ➘
    confirm("Click OK to break completely out of the loop. ➘
  \nClick Cancel to break the inner part of the loop.")

    inner :
      if (innerBoolean == true)  {
        break  outer;
      }
      else {
        break  inner;
      }
        alert("You broke the section labeled inner.")
  }
}

  if(i == 4) {
    alert("You broke the section labeled outer because the ➘
loop ended naturally.");
   }
   else {
     alert("You broke the section labeled outer.");
   }
```

In this example, the user has a choice of breaking out of the section labeled `outer` or the section labeled `inner`. Note that the labels are defined just above their associated section and are followed by a colon (`:`). The labels are then placed after the `break` keyword when the intention is to break out of that particular labeled section of code.

Instructor Note 4-4

Location: The switch *Statement*

Students may be tempted to create a `switch` statement similar to the following:

```
var aChoice = "test value";
switch(aChoice) {
  case "a value" || "another value"  :
    //code to execute
    break;
...
}
```

Note that the logical OR operator (`||`) cannot be used to create two separate conditions in a `switch` statement as shown in this example. Each test expression needs to have its own `case` statement in order for the `switch` statement to function properly.

Optional Lab 4-1: Using a *switch* statement

This optional lab will provide an opportunity to use a `switch` statement.

1. **Editor:** Open **optionalLab4-1.htm** from the Lesson 4 folder of the Student_Files directory.

2. **Editor:** A function named `switchTest()` has been started for you. A variable named `choice` is defined and receives its value from a drop-down menu selection.

3. **Editor:** Create a `switch` statement. The test expression is the `choice` variable. The text after the `case` keywords is the text from the last three drop-down menu items, **Choice 1**, **Choice 2** and **Choice 3**. Create an alert dialog box that reflects back to the user the item selected, as code to execute after each `case` statement.

4. **Editor:** Add a `default` clause that outputs an alert dialog box asking the user to make a selection from the drop-down menu.

5. **Editor:** Save **optionalLab4-1.htm**.

6. **Browser:** Open **optionalLab4-1.htm**. Your screen should resemble Figure OL4-1.

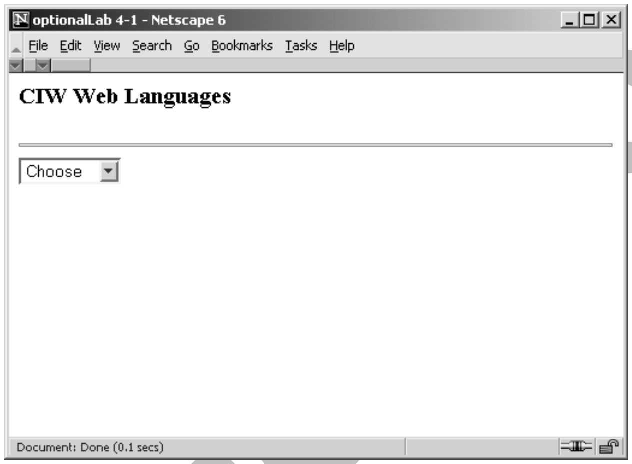
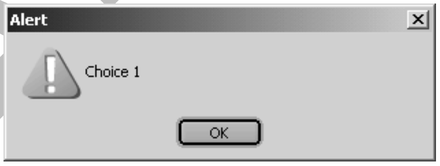

optionalLab 4-1 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
Choose
Document: Done (0.1 secs)
CIW Web Languages

Figure OL4-1: OptionalLab4-1.htm

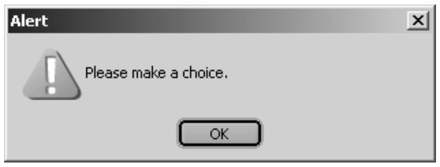
7. **Browser:** Select an item from the drop-down menu. You should see an alert dialog box similar to Figure OL4-2, depending on the selection.



Choice 1

Figure OL4-2: Alert dialog box

8. **Browser:** Click **OK** to close the alert dialog box. Select the first drop-down menu item. This action should trigger the `default` clause in your code. An alert similar to Figure OL4-3 should appear.



Please make a choice.

Figure OL4-3: Alert dialog box

9. **Browser:** Test the page to ensure that the proper alert dialog box appears for all situations.

10. **Editor:** Open **optionalLab4-1.1.htm** from the Lesson 4 folder of the Student_Files directory. This file will provide an opportunity to create a `switch` statement that executes the same code for two different `case` expressions.

11. **Editor:** A function named `switchTest()` has been started for you. A variable named `choice` is defined and receives its value from a drop-down menu selection. As in optionalLab4-1.htm, create a `switch` statement. The test expression is the `choice` variable. The text after the `case` keywords is the text from the last four drop-down menu items, **Los Angeles**, **San Francisco**, **Chicago** and **New York City**.

12. **Editor:** As the code to execute for each `case` statement, create an alert dialog box that displays for the user the state that each city is in. These states are **California** for both Los Angeles and San Francisco, **Illinois** for Chicago and **New York** for New York City. Because the same code will execute for Los Angeles and San Francisco, create a fall-through effect for those two `case` statements.

13. **Editor:** As in optionalLab4-1.htm, create a `default` clause that outputs an alert dialog box that asks the user to make a choice.

14. **Editor:** Save **optionalLab4-1.1.htm**.

15. **Browser:** Open **optionalLab4-1.1.htm**. Your screen should resemble Figure OL4-4.

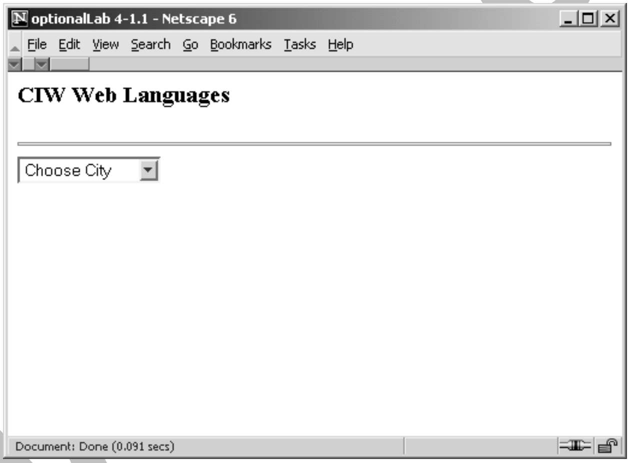

optionalLab 4-1.1 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
CIW Web Languages
Choose City
Document: Done (0.091 secs)

Figure OL4-4: OptionalLab4-1.1.htm

16. **Browser:** Select **Chicago** from the drop-down menu. An alert dialog box similar to Figure OL4-5 should appear.

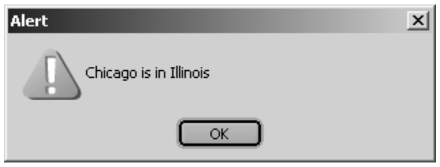


Figure OL4-5: Alert dialog box

17. **Browser:** Test the page to ensure that the proper alert dialog box appears for each situation.

In this lab, you used the `switch` statement. You learned that the `switch` construct is an effective way to create code that executes based on the comparison of one expression against many expressions. You also learned how to create a fall-through effect in a `switch` statement, so that the same code will execute for different values.

Lesson 4 Quiz

1. Which JavaScript statement should you use to force the flow of control back to the top of a loop?

   a. `break`
   b. *`continue`*
   c. `switch`
   d. `do...while`

2. Which JavaScript statement should you use to execute a block of code for as long as a certain test condition is true, with the guarantee that the code block will execute at least once?

   a. `do`
   b. *`do...while`*
   c. `if`
   d. `if...else`

3. Which JavaScript statement should you use to compare a value against other values when searching for a match?

   a. `while`
   b. `if...else`
   c. `for`
   d. *`switch`*

4. Which JavaScript statement should you use to branch to one of two processes depending on the result of a test condition?

   a. *`if...else`*
   b. `for`
   c. `break`
   d. `do...while`

5. Which JavaScript statement should you use to execute a block of code for as long as a certain test condition is true?

   a. `if`
   b. `continue`
   c. `switch`
   d. *`while`*

6. Which JavaScript statement should you use to test for a single condition and branch to a process?

   a. `while`
   b. `if...else`
   c. `for`
   d. *`if`*

7. Which JavaScript statement should you use to exit a loop that would otherwise continue to execute?

   a. `if`
   b. `continue`
   c. *`break`*
   d. `switch`

8. Which JavaScript statement should you use to repeat a group of statements for some particular range of values?

   a. `if...else`
   b. `continue`
   c. *`for`*
   d. `do...while`

9. Write an `if` statement that checks a variable named `grade` for a range of values between 70 and 80. If the value is in this range, output the following text: *Your grade is a C.*

   One way to write this statement is as follows:

```
if (grade >= 70) {
  if (grade <= 80) {
    document.write("Your grade is a C");
  }
}
```

   A better way to write this statement is as follows:

```
if (grade >= 70 && grade <= 80) {
  document.write("Your grade is a C");
}
```

10. Consider the following code:

```
function calculateAvg(a, b, c) {
  var x = parseInt(a);
  var y = parseInt(b);
  var z = parseInt(c);

  return (x + y + z) / 3;
}
```

   Write an `if...else` statement that calls the `calculateAvg()` function, then outputs one of the following: *The average is 20 or less* or *The average is greater than 20.*

```
if (calculateAvg(10, 20, 30) >= 20) {
  document.write("The average is 20 or less");
} else {
  document.write("The average is greater than 20");
}
```

11. Write a `while` loop that outputs the even numbers between 10 and 20.

```
var x = 10;
while (x <= 20) {
  if ((x % 2) == 0) {
    document.write(x + "<BR>");
  }
  x++;
}
```

12. Write a `for` loop that outputs the following:

```
10
9
8
7
6
5
4
3
2
1
0
```

```
for (var i = 10; i >= 0; i--) {
  document.write(i + "<BR>");
}
```

13. Write a `for` loop that adds 10 numbers received from user input. Offer the user a chance to exit the program using the `break` statement.

One way to write this loop is as follows:

```
var userInput = 0;
var total = 0;
for (x = 1; x <= 10; x++){
  userInput = prompt("Enter a number or s to stop.", "");
  if (userInput == "s") break;
  total += parseInt(userInput);
}
document.write("The total is " + total)
```

14. Write a JavaScript program that asks the user to guess a number between one and five, then uses a `switch` statement to evaluate the user's input. Use three as the number to be guessed. Use a `default` clause to catch input that is not in the required range.

```
var userInput = parseInt(prompt("Guess a number between 1 and 5.",
""));
switch (userInput) {
  case 1 :
    alert("Too low.");
    break;
  case 2 :
    alert("Too low.");
    break;
  case 3 :
    alert("Congratulations!");
    break;
  case 4 :
    alert("Too high.");
    break;
  case 5 :
    alert("Too high.");
    break;
  default :
    alert("Input not in required range.");
    break;
}
```

Lesson 5: The JavaScript Object Model

Objectives

By the end of the lesson, you will be able to:

- Describe the JavaScript object model.
- Use the `window` object.
- Manipulate properties and methods of the `document` object.
- Use the `with` statement.
- Deploy the `image` object.
- Use the `history` object.
- Evaluate and change URL information with the `location` object.
- Use the `navigator` object.

Pre-Assessment Questions

1. The `opener` property of the `window` object refers to:

   a. a parent frame in a frameset.
   b. *a parent window.*
   c. the location of a window.
   d. a child window of another window.

2. The `close()` method of the `document` object:

   a. closes a document in a frame.
   b. closes all cookies associated with a document.
   c. closes a document in a window.
   d. *closes the data stream to a document.*

3. Discuss the difference between the `name` property and the `src` property of the `image` object.

   The `name` property of the `image` object sets or returns the value corresponding to the NAME attribute of the <IMG> tag. The more important `src` property of the `image` object sets or returns the URL of the image. The URL information provided by the `src` property is necessary to manipulate image objects in JavaScript.

The JavaScript Object Model

JavaScript was designed explicitly for Web page use. To take advantage of the different features and capabilities provided by a browser, special browser objects have been built into the JavaScript language. These browser objects provide properties and methods to create sophisticated JavaScript programs. In addition, the JavaScript language itself provides objects that help you manipulate string, date and math information in useful ways. JavaScript also provides the `Array` object, which you will use later in this course.

The JavaScript object model divides objects into three general groups: browser objects, language objects and form field objects. Figure 5-1 shows a diagram of commonly used objects recognized by JavaScript.

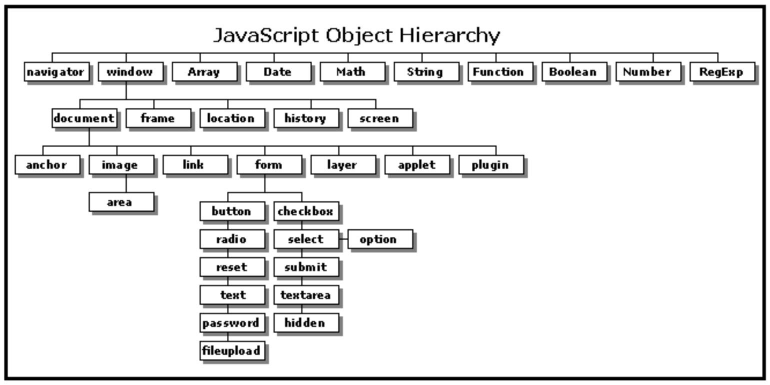


Figure 5-1: JavaScript object hierarchy

Document Object Model (DOM)
A World Wide Web Consortium specification intended to render an HTML page or XML document as a programmable object.

As mentioned, the objects in the preceding figure are a combination of different types of objects. Some objects, such as the `window`, `document` and `form` objects, represent elements of the **Document Object Model (DOM)**. The DOM is a World Wide Web Consortium (W3C) programming specification intended to give programmers access to all elements within an HTML page or XML document. Other objects, such as the `Array` and `String` objects, represent built-in JavaScript objects. These built-in objects will be discussed later in this course.

Both major browsers (Microsoft Internet Explorer and Netscape Navigator) have introduced other objects not shown in the figure. Also, neither major browser supports all the objects shown in the figure, nor do they always support the same objects in the same manner. These compatibility issues pose a serious concern to JavaScript developers. One of the challenges in creating effective JavaScript-enabled Web applications is overcoming compatibility issues between the major browsers. Due to the complexity of these compatibility issues and the time constraints of this course, we will focus on the objects commonly used by both major browsers in this lesson.

INSTRUCTOR NOTE: This section discusses the Document Object Model. Students often ask about Dynamic HTML, or DHTML. A solid understanding of the DOM is imperative when implementing DHTML. Refer student to www.w3.org/DOM/ for more information.

Containership

The hierarchy illustrated in Figure 5-1 demonstrates the principle of containership. For example, the `button` and `checkbox` objects are contained within the `form` object, which is itself contained within the `document` object, which is contained within the top-level `window` object.

Containership relates to the principle that some objects cannot be used or referenced without a reference to the parent (container) object. For example, you cannot reference a `form` element in JavaScript without referring both to the `form` object and the `document` object that contains the `form` object. To refer to the check box named `chkTennis` on a form named `myForm`, you would write `document.myForm.chkTennis`. The `window` object is the default object that contains many objects under it, thus `window` does not need to be referenced by name. The `document` object contains the `form` object, which contains the `form` elements including the `checkbox` object.

WARNING! *Note that the built-in JavaScript language objects, which are objects that do not refer to browser-related objects, are named with a capital letter. Always remember that JavaScript is case-specific. Referring to an object with improper capitalization will result in an error.*

Commonly Used Objects

In this lesson, we will discuss some of the most commonly used objects and their associated properties and methods. We will also introduce the `navigator` object in this lesson and further examine it elsewhere in the course.

The *window* Object

The `window` object is the highest-level object in the JavaScript object hierarchy, and is also the default object. The `window` object represents the frame of the browser and the mechanisms associated with it, such as scroll bars, navigation bars, menu bars and so on. Table 5-1 outlines commonly used properties, methods and event handlers associated with the `window` object.

Table 5-1: Properties, methods and event handlers of `window`

Properties	Methods	Event Handlers
`closed` Returns a Boolean value indicating whether a window is open or closed.	`alert("`*`message text`*`")` Creates a dialog box with *`message text`* as the message.	`onError` Processes when a JavaScript error occurs. (See the related appendix for more information.)
`defaultStatus` Specifies or returns a string value containing the default status bar text.	`close()` Closes the current window. You must write `window.close()` to ensure that this command is associated with a `window` object, and not some other JavaScript object.	`onLoad` Processes when the window or frame finishes loading.

Table 5-1: Properties, methods and event handlers of `window` (cont'd)

Properties	Methods	Event Handlers
`frames` Returns an array that tracks the number of frames in a window. You can target a frame in JavaScript by referencing its frame number in the `frames` array.	`confirm("`*`message text`*`")` Creates a dialog box with *`message text`* displayed, along with OK and Cancel buttons.	`onUnload` Processes when the window or frame finishes unloading.

`length`
Returns an integer value representing the number of frames in the parent window. Also returns the number of elements in the `window.frames` array.
	`moveBy(x, y)`
	`moveTo(x, y)`
	The `moveBy()` method moves a window by *`x`* pixels on the x-axis and by *`y`* pixels on the y-axis.
	The `moveTo()` method moves the top-left corner of a window to the specified position on the x-axis and y-axis.
	Both methods are available in Netscape Navigator 4.x (and later) and Microsoft Internet Explorer 4.x (and later).	

`location` Specifies or returns the `location` object for the current window.	`open("`*`URL`*`","`*`name`*`", "`*`featureList`*`")` Opens a new window, populated by *`URL`*, with the target name of *`name`*, and with the features identified in *`featureList`*.	
`name` Specifies or returns a string value containing the name of the window or frame.	`print()` Provides a script method for printing the contents of the current window and frame. Available in Netscape Navigator 4.x (and later) and Microsoft Internet Explorer 5.x (and later).	
`opener` Returns a reference to the window that called (or opened) the current window.	`prompt("`*`message text`*`", "`*`default text`*`")` Pops up a message box displaying *`message text`* with a text box for the user's response, which contains *`default text`* if not left empty. In some browsers, the word `undefined` displays in the text box if left empty.	

`parent`
Returns a string value containing the name of the parent window, if it has one.
	`resizeBy(`*`x`*`, `*`y`*`)`
	`resizeTo(`*`outerHeight`*`, `*`outerWidth`*`)`
	The *`resizeBy()`* method resizes a window by the specified *`x`* and *`y`* offsets along the x-axis and y-axis.
	The `resizeTo()` method resizes a window to the height and width specified by *`outerHeight`* and *`outerWidth`*.
	Both methods are available in Netscape Navigator 4.x (and later) and Microsoft Internet Explorer 4.x (and later).	

Table 5-1: Properties, methods and event handlers of `window` (cont'd)

Properties	Methods	Event Handlers
	`scrollBy(x, y)` `scrollTo(x, y)` The `scrollBy()` scrolls the document in a window by the specified *x* and *y* offsets along the x-axis and y-axis. The `scrollTo()` method scrolls the document in a window to the specified position on the x-axis and y-axis. Both methods are available in Netscape Navigator 4.x (and later) and Microsoft Internet Explorer 4.x (and later).	
`self` Returns a string value containing the name of the current window. Alternative for using the `name` property as previously described.	`setTimeout("`*`expression`*`",`*`time`*`)` `setTimeout(`*`functionName`*`, `*`time`*`)` Executes *`expression`* or *`functionName`* after an elapse of the interval *`time`*, an integer value representing milliseconds.	
`status` Specifies or returns a string value representing the status bar text.	`clearTimeout(`*`timerID`*`)` If the `setTimeout` to which this refers were given a *`timerID`*, this method would clear the timeout so that the related procedure would not run again when the next specified interval elapsed.	
`top` Returns a string value containing the name of the topmost window.	`blur()` Causes the window to lose focus. The window is placed beneath any other open windows.	
`window` An alternative for the `self` or `name` property.	`focus()` Causes the window to gain focus. The window is placed on top of any other open windows.	

Opening additional windows

INSTRUCTOR NOTE: This section demonstrates how to use the `open()` method of the `window` object.

JavaScript allows you to open new windows at will and populate or fill them with existing information, information created dynamically, or no information at all (a blank window). You open a new window by using the `window` object's `open()` method. As you will see in the following labs, you can launch a new window displaying an existing Web page, or you can launch an empty window and then populate it on the fly (i.e., dynamically) with content defined by your scripts.

INSTRUCTOR NOTE: The `open()` method of the `window` object provides developers with a great deal of control over the appearance of windows that an application might open. This control can be exercised in most browsers that support JavaScript without using signed scripts.

The generic syntax to open a new window using the `open()` method is as follows:

```
open("URL","WindowName","Feature List")
```

For example, to open a new browser window showing the CIW home page, you might use the following statement:

```
open("http://www.ciwcertified.com","CIWWindow", ➘
"toolbar=1,location=1,menubar=1,scrollbars=1,status=1, ➘
resizable=1");
```

Tech Note

You cannot include a space in the `WindowName` area. In the previous example, "`CIWWindow`" is valid but "`CIW Window`" would return an error in some browsers. In addition, you cannot include spaces around the = character in the window feature list options. For example, "`toolbar = 1`" will not display the toolbar, but "`toolbar=1`" will display the toolbar. Though these syntax requirements are not true of all browsers, the best policy is to adhere to them for cross-browser compatibility.

INSTRUCTOR NOTE: Refer to **Instructor Note 5-1** for more information concerning the `window` object.

The feature list consists of a series of attributes that you specify to either be displayed or not. The accepted values for each attribute are `1` or `yes` if the attribute is to be displayed, and `0` or `no` if it is not. Simply listing the attribute will also display the feature. Table 5-2 lists commonly used `window` attributes.

Table 5-2: `window` attributes accessible in `open()` method

Attribute	Description
`toolbar`	Creates the standard toolbar
`location`	Creates the Location entry field
`directories`	Creates the standard directory buttons
`status`	Creates the status bar
`menubar`	Creates the menu at the top of the window
`scrollbars`	Creates scroll bars when the document grows beyond the current window
`resizable`	Enables resizing of the window by the user
`width`	Specifies the window width in pixels
`height`	Specifies the window height in pixels
`top`	Specifies the top *y* coordinate onscreen where the window will open. Available in Microsoft Internet Explorer 4.x (and later) and Netscape Navigator 4.x (and later)
`left`	Specifies the left *x* coordinate onscreen where the window will open. Available in Microsoft Internet Explorer 4.x (and later) and Netscape Navigator 4.x (and later)

INSTRUCTOR NOTE: This section demonstrates how to use the `open()` method of the `window` object to acquire a reference to a new window. This section also demonstrates how to write dynamic content to a new window.

In some cases, you may need to open a new window and supply its content dynamically. In such cases, you must acquire a reference to the new window. The following example demonstrates the use of the `open()` method for such situations:

```
var newWindow;

newWindow = open("","dynamicWindow","toolbar=1,location=1");
```

Note that the syntax requires an empty set of parentheses where the URL would normally be placed. If this code were executed, a blank window would appear.

The open() method returns a reference to the newly opened window. In the preceding example, this reference is held in the newWindow variable. The newWindow variable can then be used to access the document object of the new window as follows:

INSTRUCTOR NOTE: Students might confuse the second attribute of the open() method with the variable name that receives the reference to a new window via the open() method. Remind students that the name created in the attribute list is never used to reference a window. It is a syntactical requirement that a name (or an empty set of quotation marks) be present when the open() method is used.

```
var newWindow;

newWindow = open("","dynamicWindow","toolbar=1,location=1");

newWindow.document.open();
newWindow.document.write("<B>Some text for the new window. </B><P>");
newWindow.document.write("<IMG SRC='myPic.jpg' HEIGHT='150' WIDTH='150'>");
newWindow.document.close();
```

Note that when using the syntax shown here, the window object cannot be implied. Recall from earlier discussions that the window object is the top-level object in the JavaScript object hierarchy, and can be omitted when calling its methods or invoking a subordinate object's methods. For example, you can write document.write() instead of window.document.write(). However, in the preceding example, you must explicitly name the window object whose document object you are accessing.

The previous example demonstrated the open() and close() methods of the document object. Do not confuse these methods with the window object's open() and close() methods. The document object's open() method prepares a document to receive a data stream. The document object's close() method closes the data stream. These topics will be discussed later in this lesson.

In the following lab, you will open a new window. This lab launches a new instance of the browser that points to another Web site.

Lab 5-1: Launching a new window with the *open()* method

INSTRUCTOR NOTE: Step 3 instructs the students to omit any height or width attributes for the first portion of this lab. Note that different browsers will handle this situation differently. The new window may open with the same size as the parent window. Or the new window may open in a frame of a different size. If the size of the new window is important, the height and width attributes should always be used.

1. **Editor:** Open **lab5-1.htm** from the Lesson 5 folder of the Student_Files directory.

2. **Editor:** Locate the existing <SCRIPT> tags. A function named newWindow() has been started for you. Inside this function, use the **open()** method of the **window** object to open a new window to the CIW Web site, using **http://www.ciwcertified.com** for the URL.

3. **Editor:** Experiment with the various attributes, such as toolbar, location and scrollbars, in the **open()** method for the new window. For this portion of the lab, omit any height or width attributes to demonstrate the effect when these attributes are not included in the open() method's arguments.

4. **Editor:** A form button is used to call the newWindow() function. This button has been scripted for you.

5. **Editor:** Save **lab5-1.htm**.

6. **Browser:** Open **lab5-1.htm**. Your screen should resemble Figure 5-2.

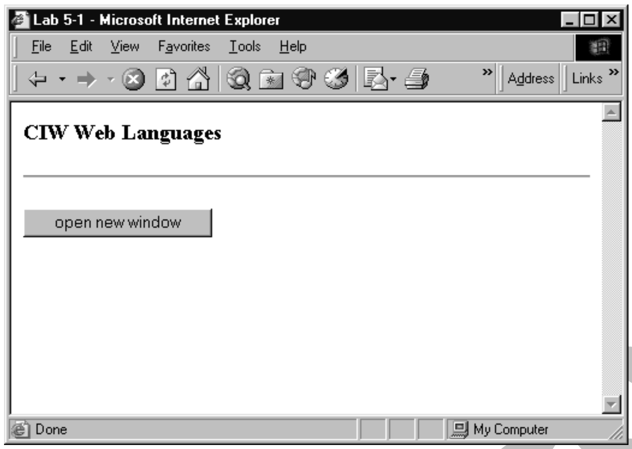

Lab 5-1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
CIW Web Languages
open new window
Done
My Computer

Figure 5-2: Lab5-1.htm

7. **Browser:** Click the **open new window** button to launch a new window. Because you did not specify a size, the new window might match the size of the existing window, or it might be a different size. The new window should resemble Figure 5-3, depending on the attributes you used in the `open()` method.

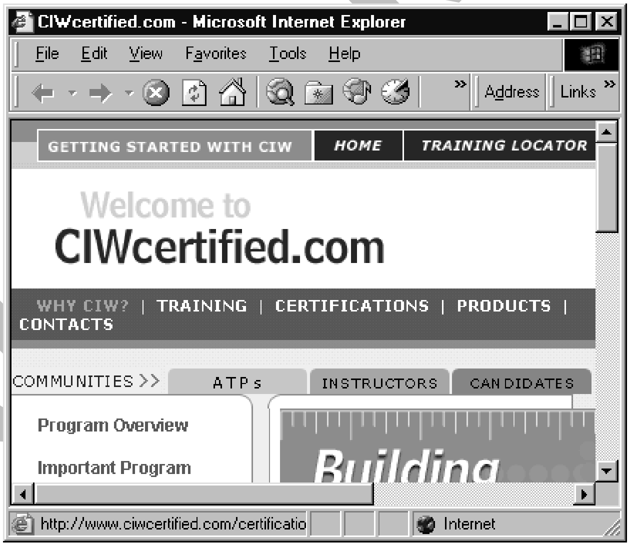

CIWcertified.com - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
GETTING STARTED WITH CIW
HOME
TRAINING LOCATOR
Welcome to
CIWcertified.com
WHY CIW? | TRAINING | CERTIFICATIONS | PRODUCTS | CONTACTS
COMMUNITIES >>
ATPs
INSTRUCTORS
CANDIDATES
Program Overview
Important Program
Building
http://www.ciwcertified.com/certificatio
Internet

Figure 5-3: CIW Web site

8. **Browser:** Close the CIW window.

9. **Editor:** Add `height` and `width` attributes to the `open()` method. Use 400 for the width and 300 for the height. Save **lab5-1.htm**.

10. **Browser:** Refresh **lab5-1.htm.** Click the **open new window** button. The new window should resemble Figure 5-4, depending on its attributes.

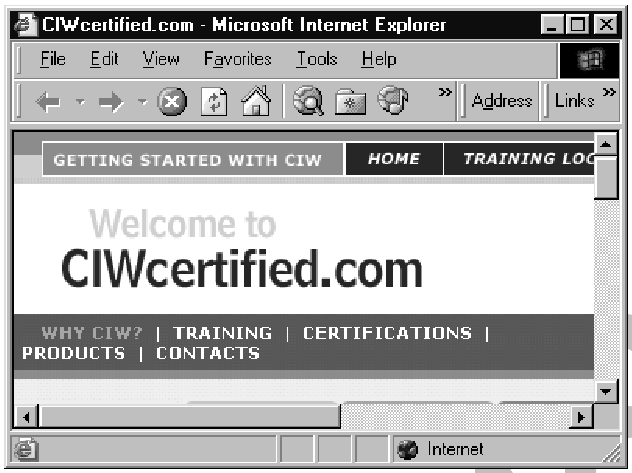

CIWcertified.com - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
GETTING STARTED WITH CIW
HOME
TRAINING LOG
Welcome to
CIWcertified.com
WHY CIW? | TRAINING | CERTIFICATIONS | PRODUCTS | CONTACTS
Internet

Figure 5-4: New window with height and width attributes

11. **Browser:** The outer frame of the new window should have a width of 400 pixels and a height of 300 pixels.

Note that browsers will not consistently interpret the height and width attributes. However, if the browser does not create a new window with the exact measurements specified, it will create a new window that is proportional to the dimensions in the attribute list.

In this lab, you used the `window` object's `open()` method to open a new window. You targeted a specific URL, which filled the new window. The function `newWindow()` was called using the `onClick` event handler of the `button` object.

You can experiment by changing some of the window attributes. For example, which actions caused the various window elements such as scroll bars and status bar to appear?

In the next lab, you will open a small window. You will assign a name to the window that will be used to refer to the new window's objects and properties. You will then write content to the new window, and learn how to close the window programmatically.

Lab 5-2: Writing content to new windows

1. **Editor:** Open **lab5-2.htm** from the Lesson 5 folder of the Student_Files directory.

2. **Editor:** Locate the existing <SCRIPT> tags. A function named **`smallWindow()`** has been started for you. Inside the function, create a variable named **`myWindow`**. Assign to `myWindow` the result of the `open()` method of the `window` object. Do not assign a URL in the `open()` method's arguments. Insert an empty set of quotation marks in the URL's place. Remember to give the window a name. For the attributes list, assign only a **`height`** of **100** and a **`width`** of **100**.

3. **Editor:** Add the `smallWindow()` function to add content to the new window. Use the reference to the new window held in the `myWindow` variable to write to the new window's document using **`document.write()`**. In the new window, create **`<FORM>`** tags. Within the form, use an **`<INPUT>`** tag to create a **`button`** object. Set the button's **`VALUE`** attribute equal to **`Close`**. Add an **`onClick`** event handler to call the **`window.close()`** method. This will provide a button for the user to close the newly opened window.

4. **Editor:** In the lab5-2.htm file, a form button is used to call the `smallWindow()` function. This button has been scripted for you.

5. **Editor:** Save **lab5-2.htm**.

6. **Browser:** Open **lab5-2.htm**. Your screen should resemble Figure 5-2.

7. **Browser:** Click the **open new window** button. You should now see a small window, as shown in Figure 5-5. If not, verify that the source code you entered is correct.

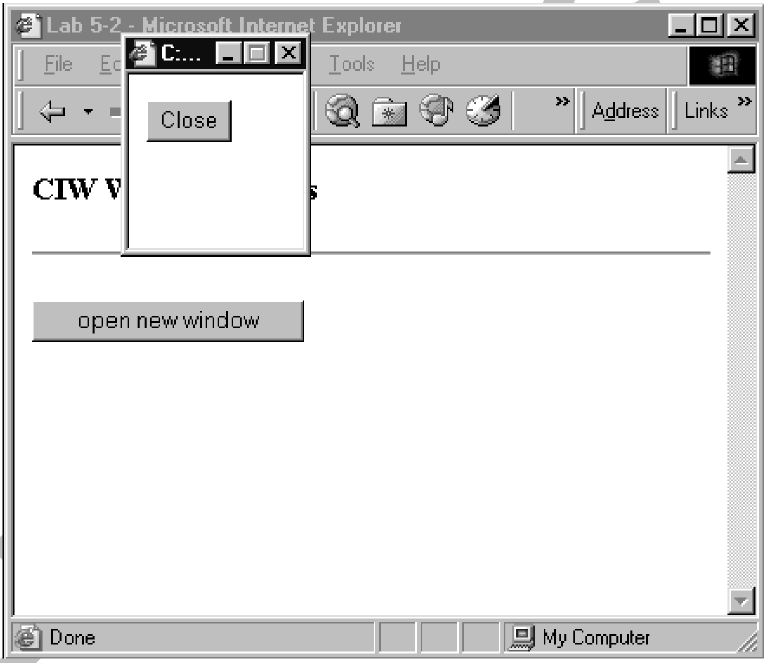


Figure 5-5: Small window opened

8. **Browser:** Click **Close** in the new window. The new window should close. If it does not, verify that the source code you entered is correct.

In this lab, you learned that you can give a window an identity. In this case, the new window was given the name `myWindow`. Because the window had a name, you could access the window's document, write to the new window, and use the `window` object's `close()` method to close that particular window. Writing dynamic content to a new window is an important concept, and will be explored further in this lesson.

WARNING! *Many Web sites today employ the `window` object's `open()` method to automatically open multiple windows for users. Be cautious when using this method because opening too many windows could discourage visitors from using your Web application.*

Dot notation revisited

INSTRUCTOR NOTE: A good analogy to use when explaining dot notation is comparing it to an Internet newsgroup address. For instance, consider the newsgroup comp.lang. javascript. The first item on the left identifies the major group. Within the comp group are numerous subgroups of which lang is one. Following the hierarchy established by comp.lang is javascript. So just as comp.lang.javascript uses dot notation to establish a hierarchy, so does `objectName.objectProperty`.

You will recall from an earlier lesson that dot notation is used to associate an object's name with the object's properties or methods. Some objects are themselves properties of a higher object. Dot notation is also used to depict this hierarchy (refer to the previous section on browser object hierarchy).

The generic syntax for dot notation is as follows:

```
objectName.objectProperty;
objectName.objectProperty = value;

objectName.objectMethod();
objectName.objectMethod(argument1,argument2,...);
```

When you need to depict a hierarchy of objects, the generic syntax is as follows:

```
parentObjectName.objectName.objectProperty;
parentObjectName.objectName.objectMethod();
```

This type of syntax was used in the preceding lab. You will use this syntax often as your experience with JavaScript grows.

The *status* property

An often-used `window` object property is `status`. The `status` property refers to the text string displayed in the status bar at the bottom of the browser window.

You can change the window's status bar text at any time during a script by using the `status` property. One common approach is to use inline scripting to make changes to the `status` property from hyperlinks and form elements. You will use inline scripting to change the `status` property in the next lab. First, however, you will learn about the `onMouseOver` and `onMouseOut` event handlers.

The *onMouseOver* and *onMouseOut* event handlers

As mentioned, inline scripting is often used to manipulate the `status` property of the `window` object. This approach is common when using the `onMouseOver` and `onMouseOut` event handlers. These event handlers are added as attributes within the HTML anchor tag, <A>, and can be added to other HTML elements as well.

When a user places the mouse pointer over the text or image associated with an <A> tag, the `onMouseOver` event handler can be invoked to respond to that event. When the user moves the mouse pointer away from a link or image, the `onMouseOut` event handler can be invoked to respond to that event as well. The following code demonstrates the syntax used for OnMouseOver and OnMouseOut event handlers:

```
<A HREF="http://www.ciwcertified.com"
 onMouseOver="status='Visit the CIW Web site.';return true;"
 onMouseOut="status='';return true;">Visit CIW</A>
```

INSTRUCTOR NOTE: This section demonstrates to the students how to use the `onMouseOver` and `onMouseOut` event handlers to manipulate the `status` property of the `window` object.

In this example, when the user places the mouse pointer over the Visit CIW link text, the status bar will read "Visit the CIW Web site." When the user moves the mouse pointer away from the text, the status bar will contain an empty string, or will revert to whatever default state it was in before the `mouseOver` event. Figure 5-6 demonstrates the execution of this example code.

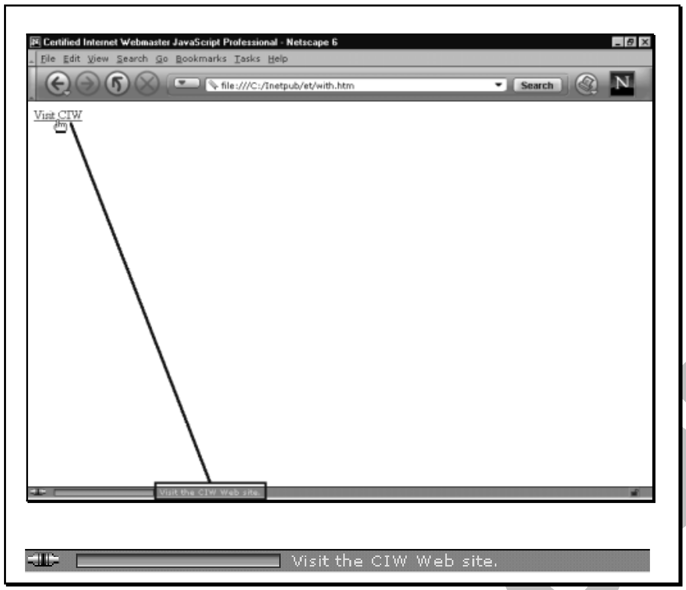

Certified Internet Webmaster JavaScript Professional - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
file:///C:/Inetpub/et/with.htm
Search
Visit CIW
Visit the CIW Web site.

Figure 5-6: Manipulating `status` property

Note that Figure 5-6 shows the Netscape 6.0 browser. As of this writing, the Netscape 6.2 browser does not display text in the status bar when executing the code shown in the previous example.

Note the use of the `return true` statement following the manipulation of the `status` property. This statement is required syntax for the `onMouseOver` and `onMouseOut` event handlers to work properly. Note the use of a semicolon (`;`) to separate the JavaScript statements. Note also that the entire `onMouseOver` and `OnMouseOut` statements are surrounded by sets of double quotation marks. The syntax for the `status` property requires quotation marks to surround the text on the right side of the equation. The syntax would be incorrect if you used another set of double quotation marks within the original set of double quotation marks. Thus, a set of single quotation marks is used, allowing the JavaScript interpreter to correctly determine where the original statement begins and ends.

INSTRUCTOR NOTE: The `onMouseOver` and `onMouseOut` event handlers functioned properly with the <A> tag in Netscape 6.0. There appears to be a bug in Netscape 6.2 that prevents manipulation of the status bar using the code shown in this section's example. No documentation or mention of this bug, or any solution for it, is posted on the Netscape Web site.

For example, if you attempted to use double quotation marks within another set of double quotation marks, your code would have the following incorrect syntax:

```
onMouseOver="status="some text";return true;"
```

The JavaScript interpreter would incorrectly assume that the opening double quotation mark was closed with the next double quotation mark encountered in the statement. Thus the statement would be interpreted as follows:

```
onMouseOver="status="
```

Obviously, this interpretation is not intended. In practice, using double quotation marks within another set of double quotation marks will cause a syntax error and prevent the code from executing properly. It is important to remember this syntax requirement when you are creating JavaScript statements of this nature.

The `onMouseOver` and `onMouseOut` event handlers are also commonly used for image swapping. This topic will be discussed later in this lesson.

In the next lab, you will change the status bar text using the `onMouseOver` event handler for the <A> tag. You will then change the status bar text to an empty string (or its default state, depending on the browser) using the `onMouseOut` event handler.

Lab 5-3: Changing status bar text

1. **Editor:** Open **lab5-3.htm** from the Lesson 5 folder of the Student_Files directory.

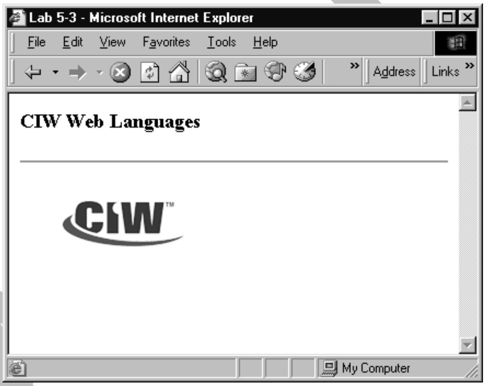
2. **Editor:** Locate the `<A HREF>` that is defined before the `<IMG SRC="images/ciw.gif">` tag. Inside the anchor tag, add an **`onMouseOver`** event handler that changes the status bar text to the following:

   `Visit the CIW Web site.`

   Also, add an **`onMouseOut`** event handler that changes the status text bar to an empty string.

3. **Editor:** Save **lab5-3.htm**.

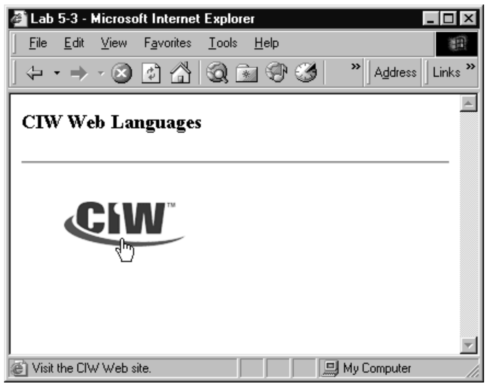
4. **Browser:** Open **lab5-3.htm**. Your screen should resemble Figure 5-7.


Lab 5-3 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
My Computer

CIW Web Languages

Figure 5-7: Lab5-3.htm

5. **Browser:** Move the mouse pointer over the CIW image. The message you created should appear in the status bar as shown in Figure 5-8. If it does not, verify that the source code you entered is correct.


Lab 5-3 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
My Computer

CIW Web Languages

Visit the CIW Web site.

Figure 5-8: Lab5-3.htm status bar message

In this lab, you learned to manipulate the `status` property of the `window` object to change the text in the window's status bar. Now that you have worked with the `window` object, you will learn about the `document` object.

The *document* Object

The `document` object is subordinate to the `window` object in the JavaScript hierarchy.

INSTRUCTOR NOTE: Most of the interaction between a user and JavaScript takes place in the `document` object. Have students review the JavaScript Object Model depicted in Figure 5-1. They will see that the `document` object provides access to most of the elements that make up an HTML page.

In any given window or frame, the `document` object is important. The `document` object provides the properties and methods to work with many aspects of the current document, including text, anchors, forms, links, page title, current location and URL, and the current background and foreground colors. The `document` object is a combination of the content and interface elements that constitute a Web page.

The `document` object is defined when the `<BODY>` tag is evaluated in an HTML page. The object exists as long as the page is loaded.

Table 5-3 outlines commonly used properties and methods associated with the `document` object. Note that some events and event handlers are associated with the `document` object, although they are not commonly used.

Table 5-3: Properties and methods of `document`

Properties	**Methods**
`alinkColor` Specifies or returns a string value representing the color for active links.	`close()` Closes a `write()` stream.
`anchors` Returns an array containing references to each anchor reference in the document. Anchor references are defined with `<A NAME="anchorName">` tags.	`open()` Opens the document to receive data from a `write()` stream.
`applets` Returns an array containing references to each <APPLET> tag in the document. Available in Netscape Navigator 3.x (and later) and Microsoft Internet Explorer 4.x (and later).	`write(`*`content`*`)` Writes the value of *`content`* to the document.
`bgColor` Specifies or returns a string value representing the background color of the document.	
`cookie` Specifies or returns a string value containing name=value pairs of data that will persist in the client's memory either until the expiration date is reached or until the browser is cleared (if no expiration date is present).	
`fgColor` Specifies or returns a string value representing the foreground color (text color) of the document.	`writeln(`*`content`*`)` Writes the value of *`content`* to the document, followed by a carriage return. Unless this command occurs within a <PRE> block of preformatted text, you will see no difference between this method and the `write()` method.

Table 5-3: Properties and methods of ***document*** *(cont'd)*

Properties	Methods
`forms` Returns an array containing references to each form in the document. A `form` object can also be referenced by name, via the NAME attribute of the <FORM> tag. Note that form elements are contained within the `form` object.	
`images` Returns an array containing references to all <IMG> tags in the document.	
`linkColor` Specifies or returns a string value representing the color of unvisited links.	
`lastModified` Returns a string value representing the date and time that the document was last modified. Note: The appearance of this string is inconsistent among browsers. Also, the availability of this property may be dependent upon server configuration.	
`links` Returns an array containing references to each link in the document. Link references are defined with <A HREF="*URL*"> tags.	
`location` String value representing the current URL. Note: This property has been replaced by the `document.URL` property in Netscape Navigator 3.x (and later) and Microsoft Internet Explorer 4.x (and later).	
`referrer` Returns a string value representing the URL of the document from which the current document was accessed. Note that `document.referrer` only returns a value if the user arrives at the current page via an HTTP link.	
`title` Specifies or returns a string value representing the text between the <TITLE> and </TITLE> tags.	
`vlinkColor` Specifies or returns a string value representing the color of visited links.	

The *bgColor* and *fgColor* properties

INSTRUCTOR NOTE: The `bgColor` property is equivalent to the `bgColor` attribute of the <BODY> tag. The `fgColor` property is equivalent to the TEXT attribute of the <BODY> tag.

Two noticeable properties of the `document` object are its background color (`bgColor`) and foreground color (`fgColor`) properties. The background color defines the color of the page itself; the foreground color defines the color of the text that displays on the page. For example, consider the following statement using an RGB color code:

```
document.bgColor = "#00FF00";
```

This statement will set or change the document's background to green.

Many browsers recognize a long list of predefined color names, such as `lightblue` and `crimson`. For example, consider the following statement:

```
document.bgColor = "black";
```

You can use this statement to define the document background color as black, or you can use the following statement with the RGB value to do the same:

```
document.bgColor= "#000000";
```

The *title* property

The `document` object has a `title` property. The `title` property holds the value of the document's title as defined between the <TITLE> and </TITLE> HTML tags. For example:

```
alert(document.title);
```

This statement would return to the user in an alert message the title of the page currently loaded. This property is read-only and cannot be set, only accessed in browsers prior to Netscape Navigator 6 and Microsoft Internet Explorer 4.

The *lastModified* property

The `document` object has a property called `lastModified` that can be used to return the date and time at which the document was most recently saved. Consider the following example:

```
document.write("This page last updated " + document.lastModified);
```

This statement will write the message "This page last updated December 17, 2001," assuming that you changed the document on that date. The appearance of the date text will vary between browsers. Also, server configuration may prevent this property from being accessed. The `lastModified` property is also a read-only property.

Referencing remote *window* and *document* objects

In addition to using and referencing the current `window` and `document` objects, you can access the methods and change the properties of objects associated with other windows. You saw a brief example of this concept in Lab 5-2. The next lab will demonstrate several `document` object properties while writing to a remote document. Before that lab, the following section will briefly discuss the `with` statement.

The *with* Statement

INSTRUCTOR NOTE: The `with` statement can be used with any valid object currently in the browser's memory. However, note that `with` statements cannot be nested.

If you are going to use several properties and/or methods with a single object, you can use the `with` statement. The `with` statement simply says that for the body of the statement, any references that follow refer to the same object. The syntax for using `with` is as follows:

```
with (objectname) {
  //statement(s) referencing objectname
}
```

INSTRUCTOR NOTE: This section demonstrates to the students how to use the `with` statement when referencing a remote `document` object.

INSTRUCTOR NOTE: Some performance degradation may be incurred if a `with` statement is used inside a loop that executes many times. This performance degradation is caused by the manner in which the JavaScript interpreter generates the artificial references to the `with` statement's object.

The `with` statement can save you some coding time. Following is an example that uses the `with` statement to add color and place text in a new window with a minimum of code typing:

```
<HTML>
<HEAD>
<TITLE>Certified Internet Webmaster JavaScript Professional
</TITLE>

<SCRIPT LANGUAGE="JavaScript">
<!--

myWindow = open("","newWin","height=100,width=100");


with (myWindow.document) {
    open();
    bgColor = "blue";
    fgColor = "white";
    write("<H2>Hello, World!</H2>");
    close();
}

//-->
</SCRIPT>
</HEAD>

<BODY>
</BODY>
</HTML>
```

If you were to load the code in this example into your browser, you would see a window similar to Figure 5-9.

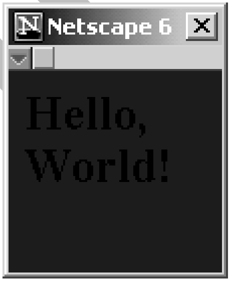


Figure 5-9: Result of using `with` statement

To illustrate how using the `with` statement can save you some typing, Table 5-4 compares the `with` statement in the last example to the same code not using the `with` statement.

Table 5-4: Comparison of code using `with` statement

Code Using *with*	Code Not Using *with*
`open()`	`myWindow.document.open()`
`bgColor = "blue"`	`myWindow.document.bgColor = "blue"`
`fgColor = "white"`	`myWindow.document.fgColor = "white"`
`write("<H2>Hello,World!</H2>")`	`myWindow.document.write("<H2>Hello,World!</H2>")`
`close()`	`myWindow.document.close()`

You will have the opportunity to use the `with` statement in the next lab.

Lab 5-4: Using properties and methods of a remote object

The file used in this lab demonstrates how to create a remote document and write dynamic content to that document. The user will be asked several questions about look and feel, as well as the content intended for the new document. After the user supplies the information, the dynamic content is written to the remote document from the script contained in lab5-4.htm.

1. **Editor:** Open **lab5-4.htm** from the Lesson 5 folder of the Student_Files directory.

2. **Editor:** Examine the `writeToDocument()` function:

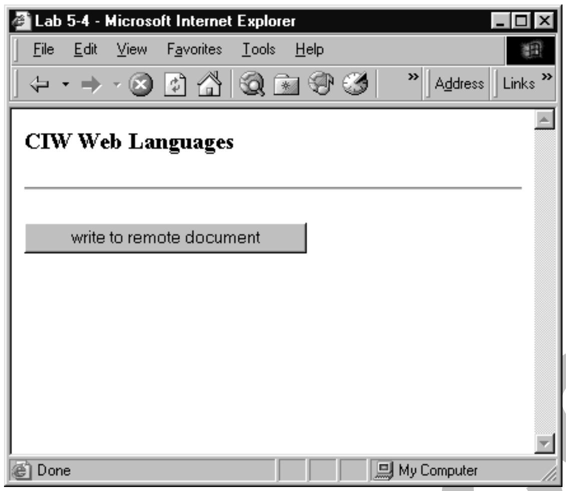
```
function writeToDocument(){
  var textColor, backColor, pageTitle;
  var yourText, pageContent, docWindow;

  textColor =prompt("Please enter a text color:","black");
  backColor =prompt("Please enter a background color:","red");
  pageTitle =prompt("The page will be titled: ", ➘
                    "Default Title");
  yourText = prompt("Add content to the new document:", ➘
                    "page content");

  pageContent = "<HTML><HEAD><TITLE>";
  pageContent += pageTitle + "</TITLE>";
  pageContent += "<SCRIPT>alert('The page: ' +  ➘
                  document.title  + ' was created: ' ➘
              +  document.lastModified);</SCRIPT>";
 pageContent += "</HEAD><BODY>";
 pageContent += "<B>" + yourText;
 pageContent += "</B></BODY></HTML>";

 docWindow = open("","docWin","width=250,height=150, ➘
                  resizable=1,status=1");

  // Create a with statement



  }
```

3. **Editor:** Note that several variables have been created. The variables `textColor`, `backColor`, `pageTitle` and `yourText` are all assigned the result of user input via `prompt()` methods. The `pageContent` variable is assigned a combination of HTML, JavaScript code, document object properties and variable values. The `docWindow` variable receives a reference to a new window via the `window.open()` method.

4. **Editor:** Locate the comment that reads **`//Create a with statement`**. Modify the **`writeToDocument()`** function to add a **`with`** statement. The `with` statement should use the **`docWindow`** document as its target object. Be sure to open the HTML data stream to the new document. Add code that will set the **`bgColor`** and the **`fgColor`**. For these, use the values in the `backColor` and `textColor` variables. Use the **`write()`** method to output the `pageContent` variable. Make sure to close the HTML data stream.

5. **Editor:** Examine the rest of the code on the page. Save **lab5-4.htm**.

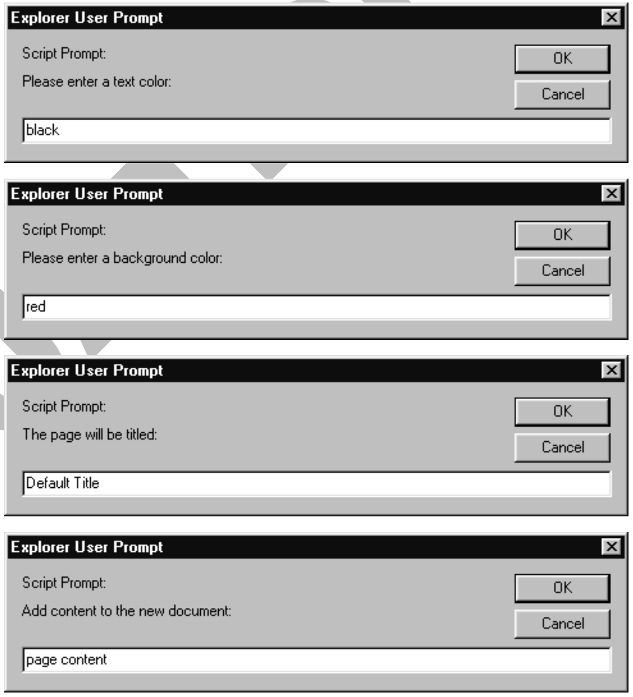
6. **Browser:** Open **lab5-4.htm**. Your screen should resemble Figure 5-10.


Lab 5-4 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
write to remote document
Done
My Computer

CIW Web Languages

Figure 5-10: Lab5-4.htm

7. **Browser:** Click the **write to remote document** button. This button has been scripted to call the `writeToDocument()` function. You will see a series of prompt dialog boxes, which will appear as you enter data (see Figure 5-11).



Script Prompt:
Please enter a text color:
black

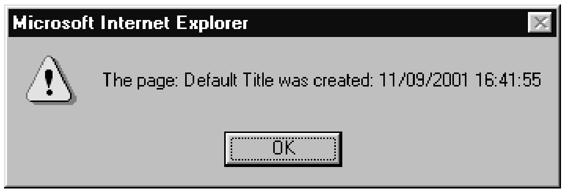
Script Prompt:
Please enter a background color:
red

Script Prompt:
The page will be titled:
Default Title

Script Prompt:
Add content to the new document:
page content

Figure 5-11: Prompt dialog boxes

8. **Browser:** After entering data into all dialog boxes and clicking **OK**, you should see an alert dialog box similar to that shown in Figure 5-12.



The page: Default Title was created: 11/09/2001 16:41:55

Figure 5-12: Alert dialog box

9. **Browser:** Click **OK**. Your screen should resemble Figure 5-13, depending on your input.

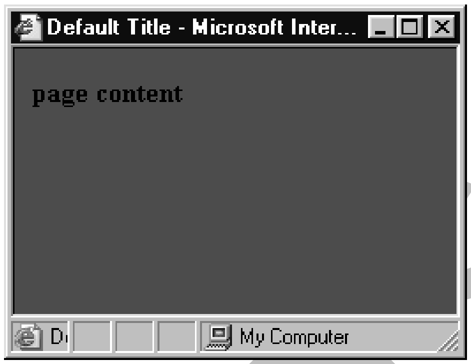


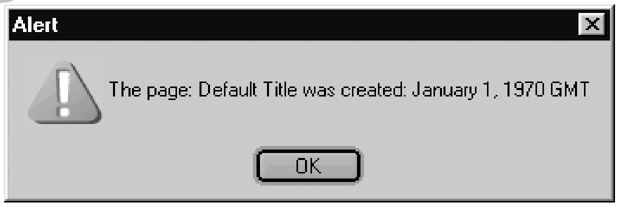
Figure 5-13: New window

INSTRUCTOR NOTE: Currently, no fix for the bug mentioned in Step 10 has been made available by Netscape.

10. **Browser:** Note that if this code were executed in Netscape Navigator 4.7, the background color would not be retained. In Netscape Navigator 6.x, neither the foreground color nor the background color would be retained. This malfunction is a bug in Netscape Navigator 4, and is in Netscape 6.x as well. A workaround for this particular problem is to pass the `backColor` and `textColor` variables directly to the appropriate attributes inside the `<BODY>` tag, as follows:

```
pageContent += "</HEAD><BODY BGCOLOR='" + backColor + "' ";
pageContent += "TEXT='" + textColor + "'>";
```

11. **Browser:** Also, Netscape Navigator does not return an accurate value for the `lastModified` property for dynamically created documents. Figure 5-14 shows the alert dialog box generated in Netscape Navigator 6.2 by lab5-4.htm.



The page: Default Title was created: January 1, 1970 GMT

Figure 5-14: Alert dialog box in Netscape Navigator

The Lesson 5 folder of the Completed_Labs directory contains a file named lab5-4.1.htm. This file contains code as shown in Step 10 of this lab.

This lab demonstrates that your scripts can reflect your HTML proficiency. The more sophisticated your knowledge of HTML, the more creative you can be when designing dynamically generated Web pages.

Several lines of code from the `writeToDocument()` function bear examination. Consider the following lines of code:

```
pageContent = "<HTML><HEAD><TITLE>";
pageContent += pageTitle + "</TITLE>";
```

These lines of code start to build the `pageContent` variable with an HTML string that will eventually be written to the new window. Note the use of the += operator in the second line of code. Using this operator is a shorthand way of expressing the following:

```
pageContent = pageContent + pageTitle + "</TITLE>";
```

In other words, the `pageContent` variable retains whatever it was already assigned and adds the new content. Note that the variable `pageTitle` is not surrounded by quotation marks because it contains variable information, whereas the literal portion of the expression (in this case `"<TITLE>"`) is inside quotation marks.

Consider the following line of code:

```
pageContent +="<SCRIPT>alert('The page: ' + document.title + ➘
' was created: ' + document.lastModified )</SCRIPT>";
```

This line of code demonstrates the fact that you can dynamically write JavaScript code to a new window. Note the concatenation of the `document.title` and `document.lastModified` properties. Also this code demonstrates again the use of single quotation marks within a set of double quotation marks. The `alert()` method's syntax requires literal strings to be inside quotation marks. Because the entire expression on the right side of the equation is inside double quotation marks, single quotation marks must be used inside the set of double quotation marks.

INSTRUCTOR NOTE: There may be bugs in the `close()` method's status bar message with some browsers on some platforms.

Finally, note that when you added the code to complete the lab, you used the `document` object's `open()` method to open a data stream to the new document. And you used the `document` object's `close()` method to close the data stream when you were finished writing to the new document. The `open()` method lets the browser prepare to accept data for display in the appropriate `document` object. The `close()` method causes the "Document Done" or "Done" message to appear in the status bar of the document's window.

The *image* Object

INSTRUCTOR NOTE: This section demonstrates to the students how to use the `image` object. The examples show how to preload images and how to dynamically swap images.

The `image` object resides beneath the `document` object in the JavaScript hierarchy. This object allows you to manipulate images in Internet Explorer 4.0 (and later) and Netscape Navigator 3.0 (and later).

One of the most popular uses of the `image` object is to create buttons that animate whenever a mouse passes over them. This function is accomplished by dynamically changing the `image` object's `src` property.

The `image` object is accessible as an array of the `document` object. An array is a group of objects or values with the same name. Each entity within this group can be accessed individually through an index number. Arrays are discussed in detail elsewhere in the book. Note that when you use the array offered by the `image` object, the name of the array is `images`, not `image`. The same applies to the `frame` object array (which is `frames`) and the `form` object array (which is `forms`). Frames and forms are discussed elsewhere in the book.

If you have an HTML document containing images, you can access the images through their array numbers using the `image` object. The standard syntax to use the `images` array is as follows:

```
document.images[0].src = newImage.jpg;
```

This statement would replace the default image (which is the first image defined in the HTML file) with `newImage.jpg`. Although the `Array` object has yet to be discussed, the preceding example demonstrates standard JavaScript syntax for accessing elements of an array. Note the array element number placed inside square brackets after the array's name. In JavaScript, the first array element is numbered `0`, the second element is numbered `1`, and so on.

Note that not all browsers support the `image` object. Whenever accessing an `image` object in your JavaScript code, ensure that the browser supports the `image` object by using an `if` statement as follows:

```
if (document.images) {
//image processing code
}
```

If the browser does not support the image object, the expression `document.images` will return a false value, and the image processing code will not execute.

The following inline scripting would replace the original image whose number is referenced by the array number with the image named `router.gif`. In this case, the second image on the page is being referenced:

```
<INPUT TYPE="button" VALUE="Change image"
onClick="document.images[1].src='/images/myImage.gif';">
```

The image can be accessed by name and by array number, as follows:

```
<IMG SRC="myPicture.gif" NAME="myImage">

<INPUT TYPE="button" VALUE="Change image"
onClick="document.images['myImage'].src='newImage.gif';">
```

In this example, the `['myImage']` portion of the statement is derived from the `NAME` attribute assigned inside the `<IMG>` tag.

You can also create an instance of the `image` object in your script. The `new` operator is used as follows:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!--
  var image1;
  image1 = new Image();

//-->
</SCRIPT>
</HEAD>
```

This syntax is important when preloading images into the browser's cache. As mentioned, one of the popular uses of the `image` object is to perform dynamic image swapping, usually using `onMouseOver` and `onMouseOut` event handlers. If the images are preloaded into the browser's cache, the user will not have to wait the few seconds to download the new image. Preloaded images improve performance of any application that uses this JavaScript functionality.

To preload images, new instances of the image object are created in the <HEAD> section of the document, as previously shown. A source for the image is then assigned using the src property. The construction of a new image object in the <HEAD> section of the document forces the browser to locate and preload that image into the browser's cache. The following example demonstrates the entire preloading procedure:

```
    <SCRIPT LANGUAGE="JavaScript">
    <!--
      if (document.images) {

        var image1, image2;

        image1 = new Image();
        image1.src = "images/image1.gif";

        image2 = new Image();
        image2.src = "images/image2.gif";
      }

    //-->
</SCRIPT>
```

INSTRUCTOR NOTE: Video source files can be preloaded by using the `Image` object's `dynsrc` property. Note that some versions of Microsoft Internet Explorer may exhibit unexpected behavior when using the `dynsrc` property.

The variables `image1` and `image2` now contain references to preloaded images available for dynamic image swapping as needed. Dynamic image swapping can be accomplished with inline scripting, as previously shown. However, functions are the most common technique used to swap images. The next lab demonstrates the use of functions to perform image swapping.

The `Image()` constructor can also receive dimension values as a parameter:

```
    image1 = new Image(150, 200);
```

The values shown are in pixels and represent the width and the height of the image.

Table 5-5 lists commonly used properties of the `image` object. The `image` object has no built-in methods.

Table 5-5: Properties and event handlers of `image` object

Properties	Event Handlers

`src`
Specifies or returns the URL of the image file. This parameter is the only one required. Following is an example of this format:
`document.images[0].src = "http://www.yoursite.com/images/comp1.gif";`
	`onAbort`
	Processes if the user selects the Stop button while an image is loading.

`height` and `width` Specifies or returns the height and width of the image in pixels. Although this parameter is not required, you should use it at all times. *Note: These properties are read-only in Netscape Navigator 3.x and 4.x.*	`onError` Processes if an error occurs while an image is loading.

`length`
Returns the number of images for a document. You can access this property using the following syntax:
`document.images.length;`	

`lowsrc` Specifies or returns an alternative low-bandwidth image.	
`complete` Returns a Boolean value indicating whether the image has loaded completely.	

Table 5-5: Properties and event handlers of `image` object (cont'd)

Properties	Event Handlers

`hspace` and `vspace`
Specifies or returns the horizontal and vertical transparent margins around the image in pixels. For example:
`document.images[0].vspace = 10;`
`document.images[0].hspace = 10;`
	`onLoad`
	Processes either when an image's `lowsrc` image finishes loading, when an image's `src` finishes loading, or when each frame of an animated GIF appears.

`border` Specifies or returns the width of the border in pixels.	
`name` Specifies or returns the value assigned to the `NAME` attribute in an `<IMG>` tag.	

Tech Note

*Although Netscape Navigator 3.0 supports the **`image`** object, Microsoft Internet Explorer 3.0 does not. Both Navigator and Internet Explorer 4.0 (and later) support the **`image`** object, but you should remember to make your code as backward-compatible as possible. If you choose not to, be sure to provide alternative means of navigation.*

Handling image object events

INSTRUCTOR NOTE: The `onAbort` event handler may be used when the user clicks the browser's Stop button while an image is loading. Note that this event handler (as well as the `onError` event handler) does not supply the name of the currently loading image. If this information is needed, the developer should store the name of the image to a variable before the image's `src` property is set.

In many applications, the <A> tag is used in conjunction with images to handle events. The <A> tag supports such event handlers as `onClick`, `onMouseOut`, and `onMouseOver`. The `image` object has several event handlers associated with it. Three of these are `onLoad`, `onError` and `onAbort`. The `onLoad` event handler activates when an image has finished loading into the browser; `onError` activates if the image itself has become corrupted or otherwise fails to load; and `onAbort` occurs when a user cancels loading of the image.

These events help you script alternatives in case a user has turned off inline images in his or her browser or a network error occurs.

In the following lab, you will use the `image` object to create an active hyperlink button. You will use the `image` object to replace a default image with another, using the `onMouseOver` and `onMouseOut` event handlers discussed elsewhere in the course.

Lab 5-5: Using the *image* object

In this lab, you will add code to preload two images for the HTML document. These images will be used to create rollover effects with each image used as a link on the page. The navigational button has two images associated with it: one for when the user places the mouse pointer over the image (on position), and one for when the user moves the mouse pointer away from the image (off position). The `src` properties will be assigned values as follows:

- `images/ciw_on.gif`
- `images/ciw_off.gif`

These images are located in the images directory of the Lesson 5 folder of the Student_Files directory.

1. **Editor:** Open the **lab5-5.htm** file from the Lesson 5 folder of the Student_Files directory.

2. **Editor:** Locate the <SCRIPT> tag in the document's <HEAD> section. Locate the comment that reads as follows:

   //Add image preloading code

3. **Editor:** Add code to ensure that the target browser supports the image object. Then add code that will preload the images provided. Create four **new Image()** objects using variables named **ciw_on**, **ciw_off**, **ciw1_on** and **ciw1_off**. Assign the **src** properties as images/ciw_on.gif for ciw_on and images/ciw_off.gif for ciw_off. Use images/ciw1_on for ciw1_on and images/ciw1_off for ciw1_off.

4. **Editor:** Your code should resemble the following (before adding the preloading code):

```
<SCRIPT LANGUAGE="JavaScript">
<!--

    //Add image preloading code







    function imageOn(imageName){
      if (document.images){
       (imageName=="ciw") ? document.images[0].src ➘
                          = ciw_on.src : "";
      (imageName=="ciw1") ? document.images[1].src ➘
                          = ciw1_on.src : "";
      }
    }

    function imageOff(imageName){
      if (document.images){
       (imageName=="ciw") ? document.images[0].src ➘
                              =  ciw_off.src : "";
      (imageName=="ciw1") ? document.images[1].src ➘
                              =  ciw1_off.src : "";
      }
    }
//-->

</SCRIPT>
```

5. **Editor:** Examine the `imageOn()` and `imageOff()` functions that have been provided for you. These functions will be discussed later in this lesson.

6. **Editor:** Scroll to the bottom of the document and locate the `<IMG SRC="images/ciw_off.gif">` tag. Note that the NAME attribute inside the <IMG> tag is set to "ciw."

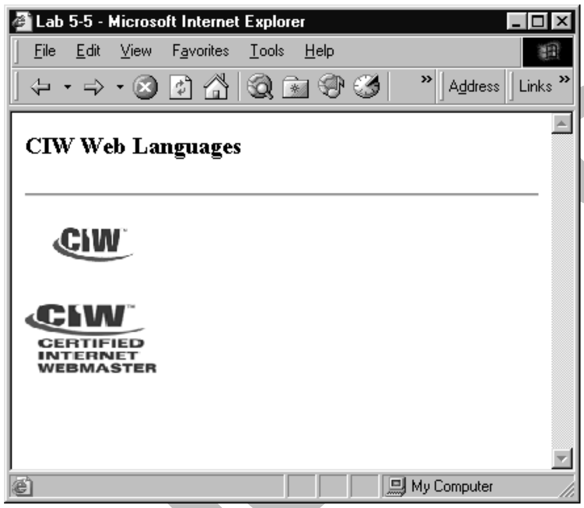
7. **Editor:** Examine the **<A>** tag that encloses the <IMG> tag. An **onMouseOver** event handler has been scripted to call the **imageOn()** function. When the function is called, the value of the NAME attribute for the <IMG> tag is passed as an argument.

8. **Editor:** Still inside the <A> tag, an **`onMouseOut`** event handler has been scripted to call the **`imageOff()`** function. When the function is called, the value of the `NAME` attribute for the `<IMG>` tag is passed as an argument.

9. **Editor:** The `<IMG SRC="images/ciw1_off.gif">` tag and its associated <A> tag are constructed in the same manner.

10. **Editor:** Save **lab5-5.htm**.

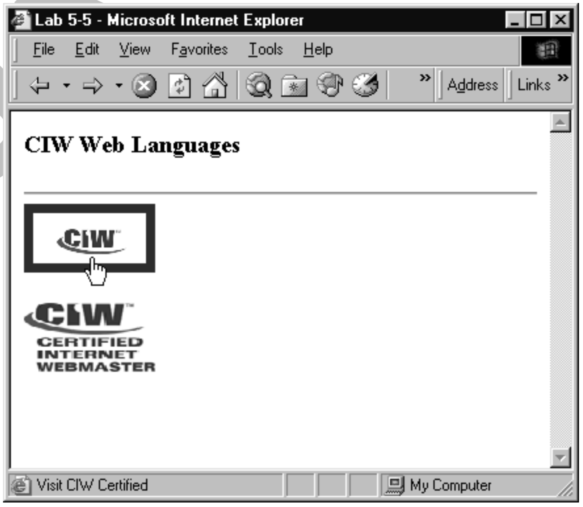
11. **Browser:** Open **lab5-5.htm**. Your screen should resemble Figure 5-15.


Lab 5-5 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
My Computer

CIW Web Languages

Figure 5-15: Lab5-5.htm

12. **Browser:** Move your cursor over the **CIW** image. Your screen should resemble Figure 5-16. If it does not, verify that the source code you entered is correct.


Lab 5-5 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
Visit CIW Certified
My Computer

CIW Web Languages

Figure 5-16: Lab5-5.htm with image swap

13. **Browser:** You should see the CIW image change when the mouse pointer passes over it, then change back to the original image when the mouse pointer is moved away. The second image on the page should function in the same manner.

14. **Editor:** If time permits, alter your code to change the `status` property of the `window` object so that it indicates the page to which the link will take the user, as shown in the preceding figure.

In this lab, you used the `image` object to create a rollover effect for navigational images on an HTML page. As you learned earlier, the best approach when creating this type of code is to preload the images into the browser's cache. You added code that created `new Image()` objects, and then used the `src` property to cause the browser to locate and preload the images into the browser's cache.

You also reviewed code that used the `onMouseOver` and `onMouseOut` event handlers to invoke the appropriate function to change the image either to the "on" position or back to the "off" position.

The `imageOn()` and `imageOff()` functions were provided for you. These functions bear examination. Consider the following code snippet from the `imageOn()` function:

```
function imageOn(imageName){
  if (document.images){
    (imageName=="ciw") ? document.images[0].src ➘
= ciw_on.src : "";
    (imageName=="ciw1") ? document.images[1].src ➘
= ciw1_on.src : "";
  }
}
```

The first line of code establishes the name of the function and readies the function to receive an argument named `imageName`. Recall that the appropriate image name is passed to the function as an argument when the function is called. The next line ensures that the browser supports the `image` object. The next line uses the conditional operator (`? :`) to determine the appropriate action to take. The first part of the statement is a Boolean expression, (`imageName=="ciw"`). If the argument passed to the function equals `ciw`, the code following the question mark will execute. The code, `document.images[0].src = ciw_on.gif`, changes the first image defined in the HTML page to the appropriate image.

If the `imageName` passed to the function is not `ciw`, the code following the colon will execute. In this case, an empty set of quotation marks indicates to the program that nothing should happen. The function is written this way so that it can be easily modified to handle multiple images on an HTML page.

The functionality of the `imageOff()` function is identical to that of the `imageOn()` function except that it changes the appropriate image back to its original state.

JavaScript and image maps

JavaScript can be used to add additional functionality to HTML image maps. See the related appendix for more information concerning JavaScript and image maps.

The *history* Object

INSTRUCTOR NOTE: In Netscape Navigator 4.0 and later, many properties of the `history` object are available only to signed scripts. This policy is for security reasons and is intended to prevent malicious scripts from accessing a user's history list. In Netscape Navigator versions before version 4.0, many of the `history` object's properties are not available at all.

The `history` object is subordinate to the `window` object in the JavaScript hierarchy.

You have already been using the same information used by the `history` object when you click the Back or Forward buttons on your browser. The browser maintains a list of recently visited Web page URLs. You can display these pages by clicking the browser buttons. If you want to give the same type of functionality to your own page, you can add similar buttons or links that allow the user to move backward or forward through the stored history of your Web page.

To access the previous pages visited, you can use the `history` object with the `back()` and `forward()` methods. The syntax is as follows:

```
<FORM>
<INPUT TYPE="button" VALUE="Back" onClick = "history.back();">

<INPUT TYPE="button" VALUE="Forward"
 onClick = "history.forward();">
</FORM>
```

Table 5-6 lists the properties and methods associated with the `history` object. Note that no events are associated with the `history` object.

Table 5-6: Properties and methods of `history`

Properties	Methods
`length` Returns an integer value representing the number of links currently referenced by the `history` object.	`back()` Sends the user to the previous page in the history list.
	`forward()` Sends the user to the subsequent page in the history list.
	`go(x)` If *x* is a negative integer, the user will be sent back *x* number of pages in the history list. If *x* is a positive integer, the user will be sent forward *x* number of pages in the history list. If *x* is a string value, the user will be sent to the page in the history list with a URL matching *x*.

INSTRUCTOR NOTE: Note that a multiple-frame window will display the parent window's `location` object information in the browser's address bar. This display does not preclude the individual frames that make up the frameset from having an associated `location` object. To access each frame's `location` object, the correct window and frame reference must be used.

The *location* Object

The `location` object is subordinate to the `window` object in the JavaScript hierarchy.

The `location` object allows you to specify URLs in a script. Thus you can create buttons instead of text or graphic links to send users to different targets. You can also tie the change of location to some other portion of script. The key property of the `location` object you will frequently use is the `href` property, which specifies or returns the hypertext reference for the current or desired URL. For example, consider the following line of code:

```
location.href = "http://www.ciwcertified.com";
```

Execution of this line in a program would send the user to the CIW Web site. You can include such a statement in a button tag, using inline scripting. The syntax is as follows:

```
<FORM>
<INPUT TYPE="button" VALUE="Visit CIW"
 onClick="location.href = 'http://www.ciwcertified.com';">
</FORM>
```

Remember that form buttons must reside within a pair of `<FORM>` tags. Note that from the beginning of `location` to the end of the tag, the script is enclosed in double quotation marks, while the URL itself is enclosed in single quotation marks.

The properties of the `location` object relate to the various pieces of information that form a complete URL. In many applications, accessing this information is extremely important. These properties allow the developer easy access to this data without having to perform rather complex string manipulation. The elements of the URL are identified as follows:

```
protocol://hostname:port/pathname/search#hash
```

Each element of the URL can be extracted from the `location` object, as indicated in Table 5-7.

Table 5-7: Properties of `location`

Property	Description
`href`	Specifies or returns the partial or full URL of a file or site. For example: `http://www.cnn.com` or `page2.htm` or `page2.htm#Section2`
`protocol`	Specifies or returns HTTP, FTP or the protocol currently in use.
`host`	Specifies or returns the *`hostname:port`* portion of the URL.
`hostname`	Specifies or returns the host name of the URL.
`port`	Specifies or returns the port number used for access (if one is provided).
`pathname`	Specifies or returns the path to a file.
`search`	Specifies or returns the text following the ? character (if a search string is present).
`hash`	Specifies or returns the internal link anchor name, which follows the hash symbol (#) in the URL.

Combining the *location* object with the conditional assignment operator

INSTRUCTOR NOTE: This section demonstrates to the students how to use the `location` object in conjunction with the conditional operator. The students may apply this information in an optional lab.

Elsewhere in this course, you learned about the conditional assignment operator, which returns one value if the specified condition is true and a different value if the condition is false. You can combine this operator with the `location` object as follows:

```
location.href = (condition) ? URL if true : URL if false;
```

This expression will result in the user being taken to one of two locations, depending on the state of the condition.

In an optional lab for this lesson, you will have an opportunity to use the `location` object.

The *navigator* Object

INSTRUCTOR NOTE: This section demonstrates to the students how to use the `navigator` object. The students may apply this information in an optional lab.

The `navigator` object reflects information about the browser being used. This object can determine the brand and version of the browser in use, and even identify the user's operating system. For example, the following line would assign a value to the `userBrowser` variable containing the name of the client (that is, `Netscape` for Netscape Navigator, and so forth):

```
var userBrowser = navigator.appName;
```

INSTRUCTOR NOTE: The information returned by some properties of the `navigator` object often can be misleading. For instance, in Microsoft Internet Explorer 5.0 for Windows, the `navigator.appVersion` property returns 4. The fact that IE 5.0 is in use can still be detected by looking for MSIE 5 in the return value of the `navigator.userAgent` property. Stress to students the importance of carefully creating and checking code that is intended to detect the browser in use.

Table 5-8 shows the `navigator` object properties and methods.

Table 5-8: Properties and methods of `navigator`

Properties	Methods
`appCodeName` Returns a string value containing the code name for the client (i.e., `Mozilla` for Navigator and `Microsoft Internet Explorer` for Internet Explorer.).	`JavaEnabled()` Returns true if Java is enabled by the client; otherwise returns false.
`appName` Returns a string value containing the name of the client (i.e., `Netscape` for Navigator and `Microsoft Internet Explorer` for Internet Explorer).	`preference()` Reads and sets user preferences on a browser (signed scripts only). Available only in Netscape Navigator 4 (and later).
`appVersion` Returns a string value containing the version information for the client.	`taintEnabled()` Returns true if data tainting is enabled by the client; otherwise returns false. (*See the following Tech Note.*)
`language` Returns a code indicating the language of the client browser. Available only in Netscape Navigator 4 (and later).	
`mimeTypes` Returns an array representing all the MIME types supported by the client.	
`platform` Returns a string that identifies the operating system for which the browser was compiled.	

Table 5-8: Properties and methods of `navigator` (cont'd)

Properties	Methods
`plugins` Returns an array representing all the plug-ins installed on a particular browser.	
`userAgent` Returns a string value containing the complete value of the user-agent header sent in the HTTP request.	

Data tainting is a Netscape Navigator 3 security feature. This feature is used to prevent scripts from accessing sensitive user information, such as directory structures and session history information. Netscape Navigator 4 (and later) replaced data tainting with signed scripts. Using signed scripts, developers can add code that will ask the user's permission before performing any operation that might be seen as a security risk.

Any of the values for the `navigator` object's properties can be accessed through a script. An optional lab for this lesson will provide an opportunity to use the `navigator` object.

Lesson Summary

Application project

This project will challenge you to use some of what you have learned in this lesson.

This application will present the user with a drop-down menu from which names of images can be selected. The image will appear in a new window for the user to preview. The application will require that your script detect the selection made by the user from the drop-down menu. As this course has not yet covered this topic, a template has been created that contains enough code to get you started. This template, named lesson5AppProj.htm, is located in the Lesson 5 Application Project folder inside the Lesson 5 folder of the Student_Files directory.

The lesson5AppProj.htm file is an HTML page with a `<SCRIPT>` block in the `<HEAD>` section of the document. A function named `display_image()` has been started for you. Two variables have been created that are assigned the text (`selectionName`) and the value (`selection`) from the drop-down menu.

Your task is to write code that will create a new window, large enough to display an image. After you have the code working properly, you can adjust the size of the window. The window will have its content dynamically written to it using `document.write()` statements.

Remember to open and close the data stream to the new window at the appropriate junctures in the code. With the new window, write code that will produce a well-formed HTML document. In other words, include `<HTML>`, `<HEAD>`, `<TITLE>` and `<BODY>` tags. Give the new document an appropriate title. Use the appropriate attributes inside the `<BODY>` tag for background and foreground colors of your choice.

In the new document, create code that will center all the content in the page. Create an `<H3>` heading, then output the value of the `selectionName` variable, creating an appropriate title for each image.

Create an <IMG> tag with the SRC attribute equal to the selection variable. The selection variable contains the path to the image file. Include HEIGHT and WIDTH attributes inside the <IMG> tag. Use 150 as the value for each of these attributes

Create an HTML form in the new document. Within the form, include a button that is scripted to close the new window.

Use the window object's focus() method to ensure that the new window remains visible when the user selects another image to preview. This code can be placed directly after the code placement in which the data stream is closed to the new document.

After you have the code working to your satisfaction, experiment with various techniques for presenting the images.

The <SELECT> tag defined in lesson5AppProj.htm uses the onChange event handler to invoke the display_Image() function. Also, the selectedIndex property of the select object is set to 0 after each selection is made. This code resets the drop-down menu in anticipation of the next user selection. These concepts are discussed in detail elsewhere in this course.

Skills review

The JavaScript object model divides objects into three general groups: browser objects, language objects and form field objects. The window object is the highest-level object in the JavaScript object hierarchy; it provides properties, methods and event handlers to work with window objects. The document object provides the properties and methods to work with many aspects of the current document, including text, anchors, forms, links, page title, current location and URL, and current colors. The image object resides beneath the document object in the JavaScript hierarchy; this object allows you to manipulate images in Microsoft Internet Explorer 4.0 (and later) and Netscape Navigator 3.0 (and later). The history object provides access to the list of URLs of recently visited Web pages maintained by the browser. The location object allows you to specify URLs in a script and provides access to all elements contained in a URL. The navigator object reflects information about the browser being used. This object can determine the brand and version of the browser in use, as well as identify the user's operating system.

Now that you have completed this lesson, you should be able to:

- ✓ Describe the JavaScript object model.
- ✓ Use the window object.
- ✓ Manipulate properties and methods of the document object.
- ✓ Use the with statement.
- ✓ Deploy the image object.
- ✓ Use the history object.
- ✓ Evaluate and change URL information with the location object.
- ✓ Use the navigator object.

Lesson 5 Review

1. What is the JavaScript object model?

   The JavaScript language provides objects that help you manipulate information in useful ways. The JavaScript object model divides these objects into three general groups (browser objects, language objects and form field objects) and organizes them into a hierarchy.

2. What is the Document Object Model (DOM)?

   The DOM is a W3C programming specification intended to render an HTML page or XML document as a programmable object, giving programmers access to all elements within such a document.

3. Describe the principle of containership. How does it apply to the JavaScript object model?

   Containership means that some objects are contained within a "parent" object, which is in turn contained within a higher-level object. This principle is illustrated in the hierarchical JavaScript object model. Some objects cannot be used or referenced without referring to the parent (container) object.

4. What is the default object? Must it be referenced by name?

   The default object is the `window` object. It does not need to be referenced by name when calling methods or invoking a subordinate object's methods.

5. What is the purpose of the `window` object's `status` property? How can you use inline scripting to manipulate it?

   The `status` property refers to the text string that appears in the status bar at the bottom of the browser window. You can change the status bar text anytime during a script. Inline scripting allows you to change status bar text from hyperlinks and form elements in response to events by using event handlers.

6. When is the `document` object defined, and for how long does it exist?

   The document object is defined when the <BODY> tag is evaluated in an HTML page. The object exists as long as the page is loaded.

7. What is the purpose of the `with` statement?

   The `with` statement allows you to use several properties and/or methods with a single object by communicating that, for the body of the statement, any references that follow refer to the same object. Using the `with` statement can save you time coding.

8. What is the purpose of the `image` object? What is one of its more popular uses?
   The `image` object allows you to manipulate images in Internet Explorer 4.0 (and later) and Netscape Navigator 3.0 (and later). One popular use is to create buttons that animate when a mouse passes over them, by using the `image` object's `src` property to swap images.

9. In what two ways can you access an image when using the JavaScript `image` object?
   By the image's name or the image's array number.

10. What is the purpose of the `history` object?
    The `history` object provides access to the list of recently visited URLs that is maintained by the browser.

11. What is the purpose of the `location` object?
    The `location` object allows you to specify URLs in a script and provides access to all elements contained in a URL.

12. What is the purpose of the `navigator` object?
    The `navigator` object reflects information about the browser being used. It can determine both the brand and version of the user's browser, as well as identify the user's operating system.

Lesson 5 Instructor Section

This section is a supplement containing additional tasks for students to complete in conjunction with the lesson. It also contains additional instructor notes. The instructor may use all, some or none of these additional tools, as appropriate to the specific learning environment. These elements are:

- **Additional Instructor Notes**
  Detailed instructor notes that expand the tips and suggestions presented in the instructor margin notes throughout the lesson.
- **Optional Labs**
  Computer-based labs to be completed during class or as homework.
- **Lesson Quiz**
  Multiple-choice test to assess student knowledge of lesson material.

Additional Instructor Notes

The following section contains detailed instructor notes that expand the tips and suggestions presented in the instructor margin notes throughout the lesson.

Instructor Note 5-1

Location: The window *Object*

JavaScript provides other attributes that can be used to manipulate a new window, when using the window object's `open()` method, that are not discussed in this lesson. Some of these attributes are specific to Netscape Navigator and some are specific to Microsoft Internet Explorer. For instance, Netscape Navigator 4.0 and later recognizes the `screenX` and `screenY` attributes to position a new window on the screen. Microsoft Internet Explorer 4.0 and later recognizes the `left` and `top` attributes to accomplish the same objective. Note that these attributes can be added to the same code because the browser will ignore any attribute it does not support. The following example demonstrates this concept:

```
newWindow = open("myPage.htm","myWindow","toolbar=1,location=1, ➘
screenX=10,screenY=10,left=10,top=10");
```

This is one example of achieving cross-browser compatibility within the same line of code.

Optional Lab 5-1: Using the *location* object with a condition

In this optional lab, you will work with a script you used previously. Using the quiz you examined in an earlier lab, you will use the `location` object to send the user to one of two pages, depending upon the number of correct answers given.

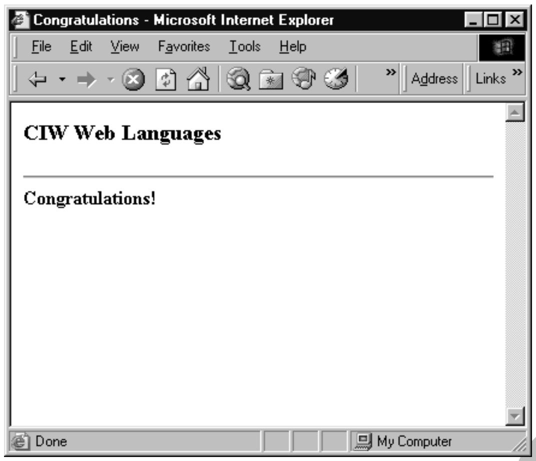
1. **Editor:** Open **optionalLab5-1.htm** from the Lesson 5 folder of the Student_Files directory.

2. **Editor:** Add a **`location.href`** statement and a conditional assignment operator to the code, so that three correct answers will send the user to a "congratulations" page. If users answer fewer than three questions correctly, send them to a "better luck next time" page. These HTML pages have been provided for you and are located in the Lesson 5 folder of the Student_Files directory. The file names are **congrats.htm** and **ohwell.htm**. These files will provide the values for the `location.href` statements.

3. **Editor:** If you are unsure where to place the `location.href` code, consider the point where it would be logical to send a user to another page. For example, would it be after the first prompt? After the first alert? Or perhaps after the third alert?

4. **Editor:** Save **optionalLab5-1.htm**.

5. **Browser:** Open **optionalLab5-1.htm**. Respond correctly to the quiz questions. You should be taken to the page shown in Figure OL5-1.

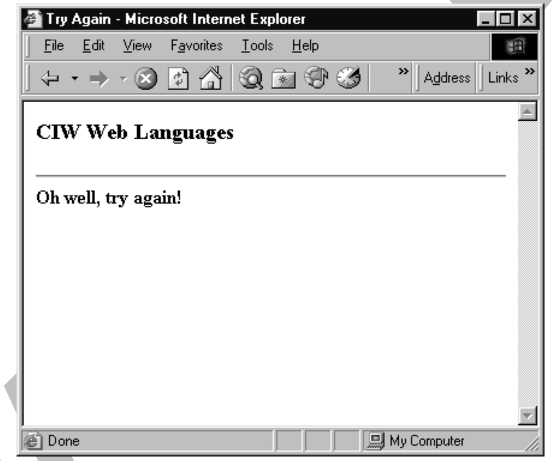

Congratulations - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
Done
My Computer

CIW Web Languages

Congratulations!

Figure OL5-1: Congrats.htm

6. **Browser:** Return to **optionalLab5-1.htm**. Answer at least one of the questions incorrectly. You should be taken to the page shown in Figure OL5-2.


Try Again - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
Done
My Computer

CIW Web Languages

Oh well, try again!

Figure OL5-2: Ohwell.htm

This lab demonstrated the `location` object used for navigational purposes. Remember that `location` object properties also provide access to each portion of a URL. This feature can be useful for applications that need to extract information from URLs.

Optional Lab 5-2: Using the *navigator* object

The following optional lab will demonstrate the type of information you can detect from several `navigator` object properties.

1. **Editor:** Open **optionalLab5-2.htm** from the Lesson 5 folder of the Student_Files directory.

2. **Editor:** Locate the <SCRIPT> block in the <HEAD> section of the file. You will add code to the `showInfo()` function. Concatenate the **`appCodeName`**, **`appName`**, **`appVersion`** and **`userAgent`** properties into the `info` variable.

3. **Editor:** The following code shows the `showInfo()` function before your changes:

```
<SCRIPT LANGUAGE="JavaScript">
<!--

function showInfo() {
  var info="";

  info += "Welcome, " +              ; //Add appCodeName here
  info += " user!\nYou are using the ";
  info +=      + " browser,\nversion "; //Add appName here
  info +=      + ".\nYour user agent "; //Add appVersion here
  info += "information is " +        ; //Add userAgent here

  alert(info);
}

//-->
</SCRIPT>
```

4. **Editor:** Save **optionalLab5-2.htm**.

5. **Browser:** Open **optionalLab5-2.htm** in the Netscape browser. Your screen should resemble Figure OL5-3.

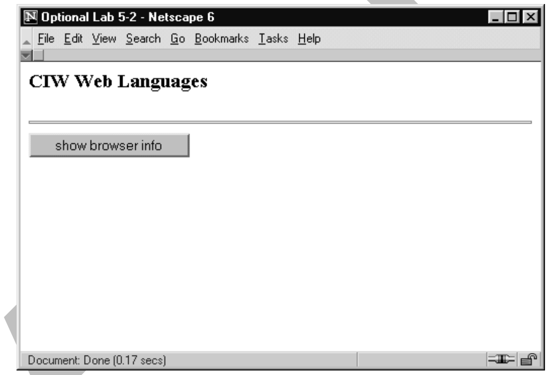
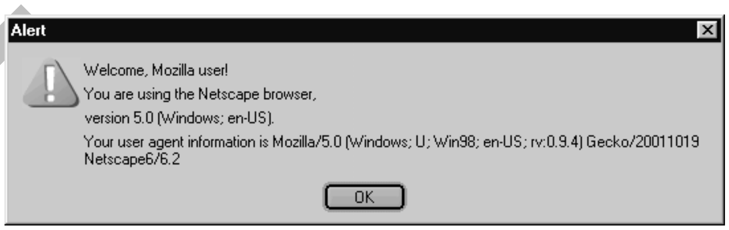

Optional Lab 5-2 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
CIW Web Languages
show browser info
Document: Done (0.17 secs)

Figure OL5-3: OptionalLab5-2.htm

6. **Browser:** Click the **show browser info** button. You should see an alert similar to Figure OL5-4.

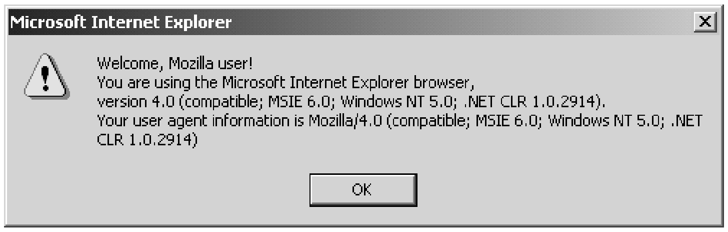

Alert

Welcome, Mozilla user!
You are using the Netscape browser,
version 5.0 (Windows; en-US).
Your user agent information is Mozilla/5.0 (Windows; U; Win98; en-US; rv:0.9.4) Gecko/20011019 Netscape6/6.2

OK

Figure OL5-4: Alert in Netscape Navigator 6.2

7. **Browser:** Open **optionalLab5-2.htm** in Microsoft Internet Explorer. You should see an alert similar to Figure OL5-5.



Welcome, Mozilla user!
You are using the Microsoft Internet Explorer browser,
version 4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.0.2914).
Your user agent information is Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.0.2914)

Figure OL5-5: Alert in Microsoft Internet Explorer 6.0

This lab demonstrated information that is available when using the `navigator` object. One of the most challenging aspects of Internet application development is creating code that will function in all browsers. When coding for different browsers is a mission-critical concern, the information that the `navigator` object contains is an essential element. As your application development experience grows, you will find many uses for the `navigator` object.

Lesson 5 Quiz

1. The JavaScript object model divides commonly used objects into what groups?
   a. Accessible elements of an HTML page or XML document
   b. *Browser objects, language objects and form field objects*
   c. `window`, `document` and `form` objects
   d. Objects supported by Microsoft Internet Explorer, Netscape Navigator, or both

2. Which JavaScript object is the default object?
   a. The `Function` object
   b. The `JavaScript` object
   c. The `String` object
   d. *The `window` object*

3. Which JavaScript object allows you to evaluate and change URL information?
   a. The `navigator` object
   b. The `history` object
   c. *The `location` object*
   d. The `link` object

4. Which JavaScript object is a combination of the content and interface elements that constitute the current Web page?
   a. The `frame` object
   b. *The `document` object*
   c. The `screen` object
   d. The `window` object

5. Which JavaScript object provides essential information if you must code your script for use in different browsers?
   a. *The `navigator` object*
   b. The `browser` object
   c. The `location` object
   d. The `history` object

6. Which JavaScript object allows you access to previously visited Web page URLs?

   a. The `location` object
   b. The `link` object
   c. The `navigator` object
   d. *The `history` object*

7. Which JavaScript object represents the frame of the browser and the mechanisms associated with it?

   a. The `document` object
   b. *The `window` object*
   c. The `frame` object
   d. The `navigator` object

8. Which JavaScript object can be used to animate a button whenever a mouse passes over it?

   a. The `button` object
   b. The `document` object
   c. The `applet` object
   d. *The `image` object*

9. Which of the following tools allows you to use several properties and/or methods with a single object?

   a. Dot notation
   b. *The `with` statement*
   c. The `write()` method
   d. The `navigator` object

10. Which general syntax should you use to open a new window with specific features using JavaScript?

    a. `open(URL,"Window Name",Feature List)`
    b. `window.open("URL","WindowName")`
    c. `open.window(URL,WindowName,Feature List)`
    d. *`open("URL","WindowName","Feature List")`*

11. Which property is associated with the `document` object?

    a. *`title`*
    b. `write()`
    c. `status`
    d. `frames`

12. Which property is commonly used with the `image` object?

    a. `open()`
    b. `location`
    c. *`src`*
    d. `onLoad`

13. Write a JavaScript statement that opens a new window that has scrollbars, menu, location, and is resizable. The new window's document will be dynamically created. Also, open the data stream to the new window's document.

```
newWin = open("", ➘
"myWindow","scrollbars=1,menu=1,location=1,resizable=1");
newWin.document.open();
```

14. What would be the size of the window opened in the previous question?

    It depends on the browser and the version of the browser in use. Normally, the new window would be the same size as the parent window.

15. Write JavaScript code that sets the background and foreground colors of a document.

```
document.bgColor = "#00FF00";
document.fgColor = "#FF0000";
```

16. Write a JavaScript statement that outputs information concerning the most recent modification of a document.

```
document.write("This document was last modified on " + ➘
document.lastModified + ".");
```

17. Write a JavaScript code block that outputs the `src` property of the various images that an HTML page might contain.

    One way to write these statements is as follows:

```
var imageNum = document.images.length;
for (var i = 0; i < imageNum; i++) {
  document.write("Image source: " + ➘
                document.images[i].src + "<BR>");
}
```

18. Write JavaScript code to preload an image named myImage.jpg that exists in the images directory of the root directory.

```
   if (document.images) {
     var newImg = new Image();
     newImg.src = "/images/myImage.jpg"
   }
```

19. A document has an image named `img1`. Create a button object that uses the `onClick` event handler to dynamically change img1 to the image that was preloaded in the previous question.

```
<FORM>
<INPUT TYPE="button" NAME="myButton" VALUE="Change Image"
onClick="document.img1.src = newImg.src">
</FORM>
```

20. Write JavaScript statements that output the protocol used to access the current HTML page.

    One way write these statements is as follows:

```
document.write("The protocol is " + location.protocol +  ".");
```

    Another way to write these statements is as follows:

```
switch (location.protocol) {
  case "http:" :
    document.write("This page access from the Web.<BR>");
    break;
  case "file:" :
    document.write("This page accessed from local computer.<BR>");
  default :
    document.write("This page not accessed from Web or local ➘
computer.<BR>");
    break;
}
```

21. Write JavaScript statements that use a confirm dialog box and the `history` object to ask users if they want to go back two pages.

```
if (confirm("Go back two pages?")) {
   history.go(-2);
}
```

22. Write a line of JavaScript code that outputs the user's browser name and version.

```
document.write("You are using " + navigator.appName + " ➘
version " + navigator.appVersion + ".");
```

Lesson 6: JavaScript Language Objects

Objectives

By the end of this lesson, you will be able to:

- Use the `String` object to test user input.
- Identify basic regular expressions and the `RegExp` object.
- Deploy the `Array` object to create more efficient code.
- Identify uses for the `Date` and `Math` objects.

Pre-Assessment Questions

1. What is the output of the following JavaScript statements?

```
var myString = "a string of text";
alert(myString.length);
```

   a. An alert dialog box displaying *a string of text* and the integer 13.
   b. *An alert dialog box displaying the integer 16.*
   c. An alert dialog box displaying *a string of text* and the integer 16.
   d. An alert dialog box displaying the integer 13.

2. What is the output of the following JavaScript statements?

```
var myString = "a string of text";
alert(myString.substring(6,3));
```

   a. An alert dialog box displaying *irts*.
   b. An alert dialog box displaying *irt*.
   c. An alert dialog box displaying *stri*.
   d. *An alert dialog box displaying tri.*

3. What is the output of the following JavaScript statements? Why?

```
var myArray = new Array();
for(i = 0; i < 10; i++) {
  myArray[i] = "value " + i + 1;
  document.write(myArray[i] + "<BR>");
}
```

The output is as follows:

value 01

value 11

value 21

value 31

value 41

value 51

value 61

value 71

value 81

value 91

A `for` loop is used to assign values to the elements of `myArray`. The loop counter variable is used as the subscript for the array's elements each time through the loop. The literal text "value" is assigned each time through the loop, with the value of the loop counter variable appended to this string. The integer 1 is then appended to the string. The code appears to be written to add 1 to the loop counter variable each time through the loop. If this was the intent, that line of code should be written as:
`myArray[i] = "value " + (i + 1);`

The output from the code would then be as follows:

value 1

value 2

value 3

value 4

value 5

value 6

value 7

value 8

value 9

value 10

Introduction to JavaScript Language Objects

In JavaScript, HTML forms and form elements are objects that can be used to manipulate string, date and math information in useful ways. For example, you can use the `String` object to manipulate and test the values in form elements before they are sent to CGI scripts and other processes.

Using the `Date` language object, you can customize a Web page on the fly, and using the `Math` object furthers your ability to create sophisticated applications. JavaScript also provides the `Array` object, which you will use elsewhere in this course. Note that the JavaScript object hierarchy model illustrated elsewhere includes the `Function`, `Boolean` and `Number` objects. Though these objects were added to JavaScript 1.2, they will not be discussed in this course. However, you will be introduced to regular expressions and the `RegExp` object.

The `String`, `Math`, `Array`, `Date` and `RegExp` objects are called language objects. They are not part of the containment hierarchy shown in the previous lesson, but are nevertheless part of the JavaScript language. First, you will work with the `String` object.

The *String* Object

INSTRUCTOR NOTE: Refer to **Instructor Note 6-1** for important information concerning some `String` object methods.

Perhaps the single most common element in any program is the string. A string can be text, numbers, or any combination of characters that functions as text. Strings can exist as literals, such as "How are you today?" or in the values of variables, such as `userName`. One of the data types in JavaScript is the `string` type, but a `String` object exists as well.

You have already seen one method of the `String` object: the `toUpperCase()` method. Any existing string in your scripts can access the methods of the JavaScript `String` object. JavaScript also offers the `new String()` constructor, which creates instances of a `String` object. A constructor is a mechanism that creates an object in JavaScript. Following is an example:

```
var myStringObject = new String("This is a string object.");

myStringObject = myStringObject.toUpperCase();
```

You could then use `String` object properties and methods with the `myStringObject` variable as shown in the preceding example. However, you do not need to create a formal `String` object to use `String` object properties and methods with the strings in your scripts. JavaScript is flexible in this regard. The following example demonstrates this concept:

```
var myString = "This is a string.";

myString = myString.toUpperCase();
```

In most cases, the syntax in the preceding example is sufficient. In other words, you normally will not need to create formal `String` objects in your scripts. However, objectifying a string is sometimes beneficial or necessary. One `String` object property—the `prototype` property—can be used only if the string has been created with the `new String()` constructor or has had the `toString()` method applied to it. The `prototype` property will be discussed later in this lesson. Also, you may need to pass an actual `String` object to certain Java applets for the string to be used properly by the applet.

The String object has methods that allow you to test for the presence of certain characters, extract a subset (called a substring) from a given string, find the position of a given character or substring in a string, and retrieve the number of characters in a string. This lesson will focus on the formatting of the text string, as well as the properties and methods that allow string manipulation.

String object formatting methods

INSTRUCTOR NOTE: This section demonstrates how to use String object formatting methods. The students will apply this information in the next lab.

The String object provides predefined methods for formatting text, which are useful when applying string display characteristics to JavaScript-generated HTML. For example, you can make a text string appear in bold type as shown in the following two examples:

```
document.write("Hello, World!".bold());
```

```
document.write(userName.bold());
```

The first example applies the bold() method to a literal string, whereas the second example applies the bold() method to a variable. The HTML generated from the first example would appear as follows:

```
<B>Hello, World!</B>
```

If desired, you can build the HTML string in the normal manner. However, the String object formatting methods can reduce the amount of typing in some situations. And the string formatting methods never neglect to close an HTML tag pair.

Table 6-1 outlines the formatting methods of the String object, with examples of their usage. Note that some of the methods require arguments to execute properly whereas others require no arguments.

Table 6-1: String object formatting methods

Method	Example	HTML Equivalent
anchor("*anchorName*")	"Part 2".anchor("p2")	<A NAME="p2">Part 2</A>
big()	"Welcome!".big()	<BIG>Welcome!</BIG>
blink()	"New today:".blink()	<BLINK>New Today:</BLINK>
bold()	"Links".bold()	<B>Links</B>
fixed()	"Name Phone".fixed()	<TT>Name Phone</TT>
fontcolor("color")	"Hi!".fontcolor("blue") "Hi!".fontcolor("#0000F")	<FONT COLOR="blue">Hi!</FONT> <FONT COLOR="#0000FF">Hi!</FONT>
fontsize(size)	"Sky Tech".fontsize(6)	<FONT SIZE=6>Sky Tech</FONT>
italics()	"Thank you.".italics()	<I>Thank you.</I>
link("URL")	"NASA".link ("http://www.nasa.gov")	<A HREF="http://www.nasa.gov"> NASA</A>
small()	"Don't look here."small()	<SMALL>Don't look here.</SMALL>
strike()	"shall grant".strike()	<STRIKE>shall grant</STRIKE>
sub()	"H" + "2".sub() + "O"	H<SUB>2</SUB>O
sup()	"E=MC" + "2".sup()	E=MC<SUP>2</SUP>
toLowerCase()	"Hello".toLowerCase()	hello
toUpperCase()	"Hello".toUpperCase()	HELLO

Note that multiple `String` object formatting methods can be added to the same string at once. Following is an example:

```
document.write("Warning".fontsize(7).anchor('warningText'));
```

Dot notation is used to add the additional methods as needed.

String object special characters

INSTRUCTOR NOTE: In typewriter terminology, a complete carriage return is a line feed (move forward one line) and a carriage return (the carriage is returned to the left margin). JavaScript treats the `\n` character as a complete carriage return. However, some CGI programs require the use of `\r\n` to perform a complete carriage return. The developer needs to be aware of the CGI program's string-handling capabilities in order to achieve specific results in certain situations.

In addition to the special formatting methods outlined in the preceding table, JavaScript provides a set of special characters that you can use to format text or simulate certain keystrokes in a script. For example, you can use a special character to create a "new line" within text: the `\n` character. Table 6-2 shows other commonly used special characters.

Table 6-2: Special characters in JavaScript

Character	Description
`\b`	Backspace
`\f`	Form feed
`\n`	New line
`\r`	Carriage return
`\t`	Tab
`\"`	Double quotation mark
`\'`	Single quotation mark
`\\`	Backslash

For example, suppose you need to put the following in a script:

```
document.write("c:\windows\webpages\test.htm");
```

INSTRUCTOR NOTE: This section demonstrates to the students how to use special characters. The students will apply this information in the next lab.

If you did not use the JavaScript special character \\ for the backslashes, the output of the preceding statement would be as follows:

```
c:windowswebpages  est.htm
```

Note that the backslashes do not appear, nor does the letter t in test. This result occurs because `\t` is a special character representing a tab character. Instead of the letter t appearing as it should, a tab space is inserted into the string. For the string to appear as intended, you must write it as follows:

```
document.write("c:\\windows\\webpages\\test.htm");
```

In the same manner, if you want quotation marks to appear in text, you must enter them with the \" special character. For example, consider the following line:

```
document.write("favorite".italics());
```

This line will display the word *favorite* on the screen in italic print.

If you want quotation marks to appear around the word, you would write the line as follows:

```
document.write("\"favorite\"".italics());
```

Now the word *"favorite"* will display on the screen in italic print with quotation marks. The safest technique is to use the JavaScript special characters when outputting double or single quotation marks or backslash characters to the screen.

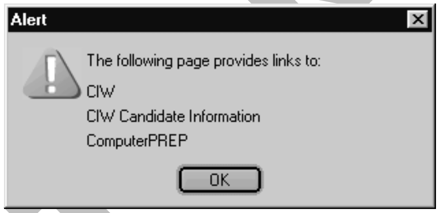
As the preceding examples demonstrate, JavaScript special characters are included with literal text. As with any other text, the special characters are placed within quotation marks when used in a JavaScript output statement.

In the next lab, you will create a new window with three linked items. You will use some of the string formatting methods shown in the previous table. You will also use one of the JavaScript special characters, the new line character, indicated by `\n`.

Lab 6-1: Using *String* object formatting methods

1. **Editor:** Open the **lab6-1.htm** file from the Lesson 6 folder of the Student_Files directory.

2. **Editor:** Locate the existing `<SCRIPT>` tag in the `<HEAD>` section of the document. Inside the existing `linksFun()` function, locate the comment that reads: **`//Add the alert() shown in Figure 6-1`**. Create an **`alert()`** method that outputs text as shown in Figure 6-1. Use the `\n` special character to create the line breaks.



```
The following page provides links to:
CIW
CIW Candidate Information
ComputerPREP
```

Figure 6-1: Alert with line breaks in text

3. **Editor:** Locate the comment that reads: **`//Add big() and fontcolor() methods`**. Add the **`big()`** and **`fontcolor()`** methods to the text **CIW Sites**. Choose any color as the argument for the **`fonctcolor()`** method.

4. **Editor:** Locate the comment that reads: **`//Add italics() method`**. Add the **`italics()`** method to the text **CIW**.

INSTRUCTOR NOTE: Refer to **Instructor Note 6-2** for information concerning the **\n** special character used in this lab.

5. **Editor:** Locate the comments that read:

```
//Add link() method
//Use http://www.ciwcertified.com.
```

Add the **`link()`** method to the text **CIW**, **CIW Candidate Information**, and **ComputerPREP**. Use the URLs included in the comments for the `link()` method arguments as shown in the following code:

```
<SCRIPT LANGUAGE="JavaScript">
<!--

function linksFun() {

var content="", linkWindow;
// Add the alert() shown in the figure



content += "<HTML><HEAD><BASE TARGET='_blank'></HEAD><BODY>";


// Add big() and fontcolor() methods

content += "CIW Sites"    ;
content += "<P>\nThese are sites of interest to ";

// Add italics() method

content += "CIW"      ;
content += " candidates.<P>\n";
content += "<UL><LI>";

// Add link() method
// Use http://www.ciwcertified.com

content += "CIW"      ;
content += "\n<LI>";

// Add link() method
// Use http://www.ciwcertified.com/introCND.asp?comm=CND

content += "CIW Candidate Information"         ;
content += "\n<LI>";

// Add link() method
// Use http://www.computerprep.com/

content += "ComputerPREP"       ;
content += "\n</UL></BODY></HTML>";

linkWindow = open("","Links","width=350,height=200,resizable=1");

    with(linkWindow.document) {
      open();
      write(content);
      close();
   }
}

// -->
</SCRIPT>
```

6. **Editor:** Examine the rest of the code in the `linksFun()` function.

7. **Editor:** In the body of the document, locate the comment that reads:

   `<!--Add HREF code -->`.

   Modify the <A> tag as indicated in bold (this code will be discussed following the lab):

```
<A HREF="javascript:void(linksFun());"
onMouseOver="status='CIW Links';return true;"
onMouseOut="status='';return true;">
<IMG SRC="images/ciw.gif" WIDTH="181" HEIGHT="84" BORDER="0"></A>
```

8. **Editor:** Save **lab6-1.htm**.

9. **Browser:** Open **lab6-1.htm**. Your screen should resemble Figure 6-2.

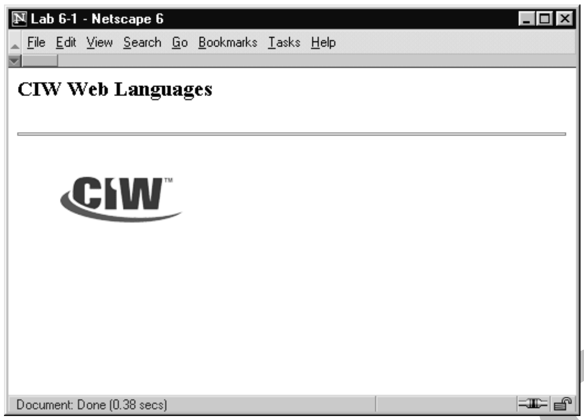
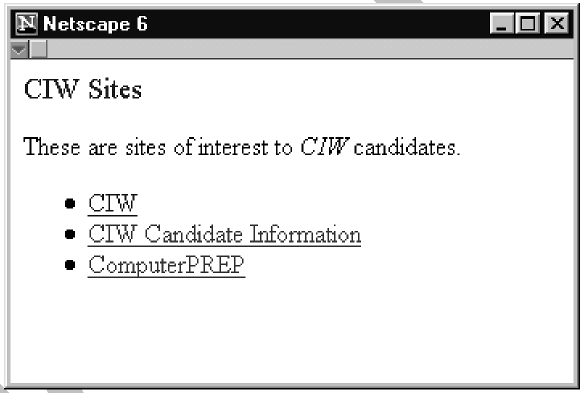

Lab 6-1 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
CIW Web Languages
CIW
Document: Done (0.38 secs)

Figure 6-2: Lab6-1.htm

10. **Browser:** Click the **CIW** image. You should first see an alert dialog box as shown in Figure 6-1. You should then see a smaller window appear. The small window should resemble Figure 6-3. If it does not, verify that the source code you entered is correct.



CIW Sites

These are sites of interest to *CIW* candidates.

- CIW
- CIW Candidate Information
- ComputerPREP

Figure 6-3: Links window

11. **Browser:** Click the links to make sure they access the appropriate sites.

In this lab, you learned how to use some of the formatting methods of the `String` object to manipulate text in a new window. As time permits, try editing the source code file to incorporate other formatting methods and other special characters.

Also in the preceding lab, you added the following line of code:

```
<A HREF="javascript:void(linksFun());">
```

The `javascript:` URL syntax can be used in any HTML tag that includes `HREF` or `SRC` attributes. This syntax allows the developer to script actions in these tags without navigating anywhere. In the preceding example, the `void` keyword prevents the link from outputting any value that an invoked function might return. The `javascript:void()` mechanism is a handy tool for calling JavaScript functions from links without sending the user to a new page.

The *prototype* property

The String object provides the powerful prototype property. This property allows you to create your own methods and properties for the String object. After you create new methods or properties with the prototype property, those methods or properties are available to any subsequent String objects in the current document.

To create a new method for the String object, define the text manipulation to be performed in a normal JavaScript function. Then use the prototype property to assign the new method to the String object. To create a new property, use the prototype property to name and define the new property. Consider the following example:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!--

  function blueVerdanaFont() {
    return "<FONT FACE='Verdana' COLOR='blue' SIZE='6'>" + ➘
this.toString() + "</FONT>";
  }
  String.prototype.blueV = blueVerdanaFont;
  String.prototype.blueVDescription = "The blueV() function ➘
creates blue Verdana font.";

//-->
</SCRIPT>
</HEAD>

<BODY>

<SCRIPT LANGUAGE="JavaScript">
<!--

  document.write("The following text is " + ➘
  "blue Verdana".blueV() + " font.");
  document.write("<P>" + "blueV".blueVDescription);

//-->
</SCRIPT>
</BODY>
</HTML>
```

INSTRUCTOR NOTE: Any JavaScript object created with the **new** constructor can utilize the prototype property to create new properties and methods for that object. In the example in this section, JavaScript applies String object methods and properties to strings whether or not they are created with the **new** constructor.

The function named blueVerdanaFont() defines the functionality of the new String method. The string data in the function is returned to a String object when the function is called as the object's method. The this keyword is a reference to the object invoking the method, and calling the method is converted to a string by the toString() method, and is concatenated with the other strings to be returned to the calling object. Note the following line of code:

```
String.prototype.blueV = blueVerdanaFont;
```

Following the function definition, this line creates the method name blueV(). Note that the name blueV() is programmer defined, and that no parentheses follow the function name blueVerdanaFont. If present, the parentheses would invoke the function, which is not this code's intended purpose. Now, just as with any other String object method, the syntax blueV() can be used to call the method. The following code from a previous example demonstrates this concept:

```
"blue Verdana".blueV()
```

The new `String` property named `blueVDescription` is created with the following code from the previous example:

```
String.prototype.blueVDescription = "The blueV() function creates blue ➘
Verdana font.";
```

The value held by this new property is then accessed with the following code:

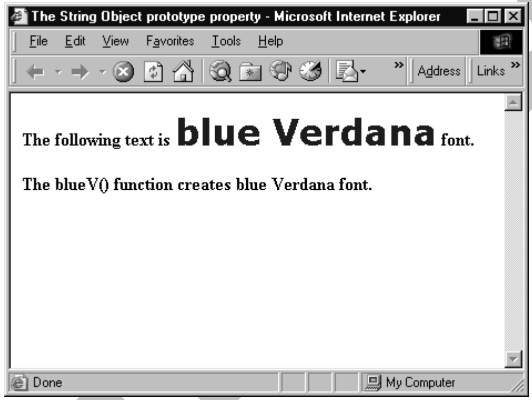
```
document.write("blueV".blueVDescription);
```

INSTRUCTOR NOTE: New `String` object methods and properties created with the `prototype` property are available to any string in the current document. A common mistake for new JavaScript programmers is to forget that the new method or property is not available throughout an entire application. The new methods or properties must be defined on any page that needs to use them.

Note that the `blueVDescription` property could also be accessed using the following syntax:

```
document.write(String.prototype.blueVDescription);
```

If the code from the example were executed, the output would resemble Figure 6-4.


The String Object prototype property - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
Done
My Computer

The following text is **blue Verdana** font.

The blueV() function creates blue Verdana font.

Figure 6-4: Output of `String` prototype demonstration

The methods that you create and attach to the `String` object can be as complex as needed. Though only string formatting was demonstrated in this example, any type of string manipulation can be performed by prototyped `String` object methods.

Additional *String* Object Methods

Thus far, this lesson has introduced you to strings and several text-formatting methods associated with the `String` object. These methods are useful and can save time, but are not the most powerful methods provided by the `String` object. Now, you will learn methods that allow you to evaluate strings for such items as length, content and the presence or absence of certain characters.

Evaluating Strings

To examine strings (possibly to validate form element values or cookies), you must first learn the available evaluation methods of the `String` object. This section will teach you to effectively examine your string data for programming purposes.

The *length* property

INSTRUCTOR NOTE: These sections demonstrate to the students how to use `String` object evaluation methods. The students will review code in the next lab that uses this information. These sections are the student's reference points if they need to see how to perform these tasks.

A common task in data validation is to test the length of a string. For example, suppose you create a form with a text box in which the user enters his or her e-mail address. Before the form is submitted, you can test whether the e-mail address has at least six characters (the minimum number of characters for a valid e-mail address). To determine the length of any string, you can use the `length` property. The syntax is as follows:

```
string.length;
```

Following is an example that tests the length of a literal string:

```
"Hello, World!".length;
```

This expression would result in the integer value of 13, representing the number of characters in the phrase "Hello, World!" including the blank space. The following example tests the length of a variable:

```
var userName = "Jose";

userName.length;
```

The second statement in the preceding example would return the integer value of 4, or the length of whatever string the `userName` variable contains.

The *indexOf()* and *lastIndexOf()* methods

The `indexOf()` method tests for the presence of a given character and returns an index value representing the character's position in the string. For example, suppose you want to test whether the user entered a valid e-mail address. You could test for the existence and position of any given character, such as the @ symbol.

The generic syntax of the `indexOf()` method is as follows:

```
string.indexOf(searchText, startingIndexPosition);
```

For example, consider the following statement:

```
"This is a test".indexOf("h", 0);
```

INSTRUCTOR NOTE: If a JavaScript program that uses the `indexOf()` or `lastIndexOf()` methods needs the exact index position of a character or character that begins a substring within a string, the developer needs to add one to the return value of the method to arrive at the correct index position.

Examine the argument portion of the statement `("h", 0)`. This argument instructs the `indexOf()` method to search the string for the first occurrence of the letter `h` starting at the character indexed zero (first character in the string). This statement would return the value of `1` because the letter `h` is the first character to the right of the beginning (that is, the second character) of the string. In JavaScript, the index for the string is zero-based. Thus the letter `T` in this example string is in position 0. Now consider the following example:

```
"johndoe@company.com".indexOf("@", 0);
```

This statement would return the value of `7`. However, consider the following statement:

```
"johndoe@company.com".indexOf("doe@", 5);
```

This statement would return the value of `-1` because no occurrence of the string `doe@` is found from position 5 (sixth from the left) to the end of the string. A value of `-1` always means the sought-after string was not found, searching forward from the specified starting position. Note that regardless of where the search starts in the string, the returned integer value is always relative to the first character in the string. If a `startingIndexPosition` is not specified, the search will always begin at the first character (position 0) in the string.

If you want to verify the proper format of an e-mail address, you could use the `indexOf("@")` expression to verify the existence of an @ symbol.

In some versions of Netscape Navigator 2 and 3, if the string being searched is empty, the `indexOf()` method may return an empty string instead of the –1 value that your scripts might expect. Check for an empty string prior to using the `indexOf()` method if these browsers might be in use by your intended users.

The method called `lastIndexOf()` is a variation on `indexOf()`. As you would assume, this method returns the index value for the starting position of the last occurrence of the search text specified. The mechanism is slightly different, as `lastIndexOf()` searches from the ending position instead of the starting position of the string. The generic syntax for `lastIndexOf()` is as follows:

```
string.lastIndexOf(searchText, startingIndexPosition);
```

For example:

```
"ABRACADABRA".lastIndexOf("ABRA", 10);
```

Examine the argument portion of the statement `("ABRA", 10)`. This argument instructs the `lastIndexOf()` method to search the string for the last occurrence of ABRA starting at the tenth, or last, character of the string. This statement would return the value of 7, even though ABRA occurs first starting at position 0. Consider the next example:

```
"ABRACADABRA".lastIndexOf("ABRA", 6);
```

The argument portion of this statement instructs the `lastIndexOf()` method to search the string starting at character position 6. This statement would return the value of 0, correctly locating the search string ABRA that starts at index position 0.

INSTRUCTOR NOTE: New JavaScript programmers often forget that the return value of the `lastIndexOf()` method is relative to the beginning, not the end of the string.

Similar to the `indexOf()` method, the `lastIndexOf()` method will start at the end of the string if no `startingIndexPosition` is specified. And the return integer value is always relative to the first character in the string.

The `lastIndexOf()` method has been prone to numerous bugs, especially in Netscape Navigator 2 and in later Navigator versions for UNIX. Test your scripts thoroughly when using the `lastIndexOf()` method.

The *substring()* method

When you want to examine and extract a portion of a string, you can use the `substring()` method. The generic syntax for the `substring()` method is as follows:

```
string.substring(startingIndexPosition, endingIndexPosition);
```

INSTRUCTOR NOTE: Refer to **Instructor Note 6-3** for information concerning the `substr()` method, which is closely related to the `substring()` method.

The `startingIndexPosition` argument is the zero-based count from the beginning of the string, and the `endingIndexPosition` argument is one character past where you want the extracted string to end.

Consider the following example:

```
"ABC1234XYZ".substring(3,7);
```

This statement would return the following value:

```
1234
```

In this statement, `1` is the fourth character (in index position 3) and `4` is the last character before index position 7. So `1234` is the sequence that fits from index position 3 up to but not including index position 7.

The `substring()` and `indexOf()` methods are often used in tandem to perform string manipulation and validation. For instance, you could determine the host name in an e-mail address as follows:

```
var myEmail = "gregory@someplace.com";
var start = myEmail.indexOf("@") + 1;
var myHost = myEmail.substring(start, myEmail.length);
alert(myHost);
```

In the preceding example, the variable `myEmail` contains an e-mail address. The variable `start` is assigned the integer value that results from the `indexOf()` method applied to the `myEmail` variable, searching for the @ symbol. Note that the integer `1` is added to this value. The expression `myEmail.indexOf("@")` returns the integer value of `7`. Because the host name is extracted from the e-mail address, you must start one position past the @ symbol. Thus the integer `1` is added to the original return value. The `myHost` variable is assigned the substring that is extracted starting at index position 8 and ending at the integer value returned by `myEmail.length`, or `21`. If you could see that line of code upon execution, it would appear as follows:

```
var myHost = myEmail.substring(8, 21);
```

In this case, the `substring()` method extracts and returns the string starting at index position 8 (the letter s in "someplace") and ending one character before index position 21 (the letter m in "com"). This extracted string is the host name, someplace.com.

INSTRUCTOR NOTE: A common mistake is to use the `substring()` method when extracting single characters from strings. Remind students that the `charAt()` method is a more efficient approach when extracting a single character from a string.

The *charAt()* method

When you want to extract a character that occurs at a specific position in a string, you can use the `charAt()` method. The generic syntax for the `charAt()` method is as follows:

```
string.charAt(indexPosition);
```

The `indexPosition` argument is an integer value that represents the index position of the character to be extracted. Consider the following example:

```
"ABCDEF".charAt(3);
```

This statement would return the value `D`.

The `charAt()` method considers the string index values to be zero-based just as with the preceding string evaluation methods. This method is similar to the `substring()` method, but `charAt()` extracts only a single character.

As mentioned earlier, the `String` object methods and properties are often used in tandem. For instance, to extract the last character in a string, you could use the following:

```
var myString = "ABCDEF";
var lastChar = myString.charAt(myString.length - 1);
```

In this example, the expression `myString.length - 1` returns the integer value `5`. If you could see that line of code upon execution, it would appear as follows:

```
var lastChar = myString.charAt(5);
```

This `lastChar` variable would contain the value `F`.

Form validation using string methods

Using string methods, you can test for certain patterns in information that a client may provide via HTML forms. For example, credit card numbers follow patterns for which you can test. Although a rigorous implementation of this technique is beyond the scope of this course, you should consider using `String` object methods in this way.

Testing the length, checking for the presence of certain characters, checking the position of substrings, and the other methods you learned in this lesson are important when performing client-side form validation.

You cannot ensure that the information a user entered is correct, but you can trap data that is undeniably incorrect. For example, if you request an e-mail address and the @ symbol is not present, you can be sure that the information the user entered is not valid e-mail address syntax. However, the presence of the @ does not guarantee that the e-mail address entered is valid or accurate; anyone can submit a false e-mail address. However, limited validation is better than no validation at all.

The following lab demonstrates some methods of the `String` object used to examine and manipulate the contents of several text boxes. In addition, this lab reviews another method: the `toUpperCase()` string conversion method. You will learn more about form validation later in the course.

Lab 6-2: Applying *String* methods to text

INSTRUCTOR NOTE: The code for this lab demonstrates how to extract a value from various `form` object elements using JavaScript. This concept is discussed in depth elsewhere in the coursebook.

1. **Editor:** Open the **lab6-2.htm** file from the Lesson 6 folder of the Student_Files directory.

2. **Editor:** Examine the following JavaScript code (this code will be discussed in detail following the lab):

```
<SCRIPT LANGUAGE="JavaScript">
<!--

function showUpper(checked) {
   var showThis;
    if (checked)  {
        showThis = document.myForm.name.value.toUpperCase();
    } else {
        showThis = document. myForm.name.value.toLowerCase();
    }
    document.myForm.name.value = showThis;
}

function emailTest(form)  {
    if (form.email_address.value.indexOf("@", 0) < 0) {
        alert("This is not a valid e-mail address!");
    } else {
        alert("This could be a valid e-mail address");
    }
}

function showFirst2(form)  {
   var first2Chars;
    first2Chars = form.phone_number.value.substring(0, 2);
    alert(first2Chars);
}
//-->
</SCRIPT>
```

3. **Editor:** Locate the <FORM> tag in the source code and examine the following (particularly the code in bold):

```
<FORM NAME="myForm">
Name:<BR>
<INPUT TYPE="text" SIZE="30" NAME="name">

<INPUT TYPE="checkbox" NAME="upperCheckbox"
onClick="showUpper(this.checked);">
Convert string to uppercase or lowercase<BR>

Email address:<BR>
<INPUT TYPE="text" SIZE="30" NAME="email_address">

<INPUT TYPE="checkbox" onClick="(this.checked) ?  emailTest(this.form) : '';">
Test for email address<BR>

Phone number:<BR>
<INPUT TYPE="text" SIZE="30" NAME="phone_number"><BR>
Fax number:<BR>
<INPUT TYPE="text" SIZE="30" NAME="fax_number"><P>

<INPUT TYPE="button"
 VALUE="first two characters of phone number"
 onClick="showFirst2(this.form);"><P>

<INPUT TYPE="button" VALUE="fax number length"
 onClick="
   var strLength = document.myForm.fax_number.value.length;
    alert('That string is ' + strLength + ' characters long');
    ">
</FORM>
```

4. **Editor:** Close the file.

5. **Browser:** Open **lab6-2.htm.** Your screen should resemble Figure 6-5. Enter some text and test your options.

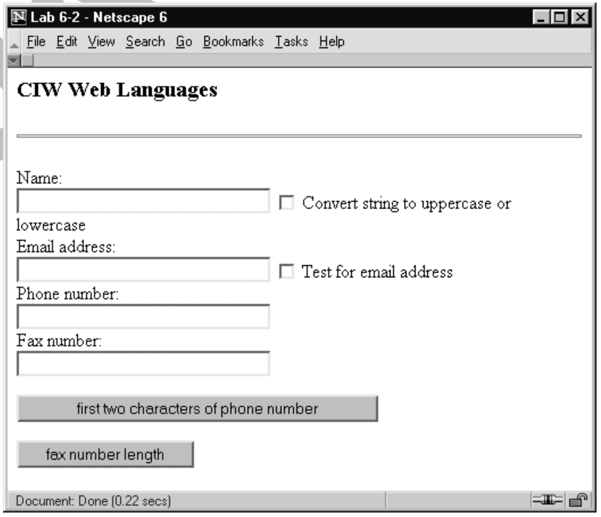

Lab 6-2 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
CIW Web Languages
Name:
Convert string to uppercase or lowercase
Email address:
Test for email address
Phone number:
Fax number:
first two characters of phone number
fax number length
Document: Done (0.22 secs)

Figure 6-5: Evaluating strings with string methods

6. **Editor:** As time permits, modify the **emailTest()** function to include a test that ensures that the e-mail address is at least six characters in length. A suggested solution named **lab6-2.1.htm** is included in the Completed_Labs\Lesson 6 directory.

This lab demonstrated several `String` object methods. In the program, the first function is named `showUpper()`. This function is called with the following code:

```
<INPUT TYPE="checkbox" onClick="showUpper(this.checked);">
```

This code uses the `button` object's `onClick` event handler to call the `showUpper()` function. The argument passed to the function is `this.checked`. The keyword `this` is used in JavaScript to refer to the object that is using it in a script. In this case, `this.checked` is a shorthand way of expressing the following:

```
document.myForm.upperCheckbox.checked;
```

The lesson on "Developing Interactive Forms with JavaScript" will discuss form elements and accessing form element values in detail. For now, know that a `checkbox` element has a Boolean property named `checked`. This property returns a `true` value if the check box is checked, and a `false` value if it is not. When the `showUpper()` function is called, it is passed either `true` or `false` depending on the state of the check box.

Following is the `showUpper()` function:

```
function showUpper(checked) {
   var showThis;
    if (checked)  {
        showThis = document.myForm.name.value.toUpperCase();
    } else {
        showThis = document. myForm.name.value.toLowerCase();
    }
    document.myForm.name.value = showThis;
}
```

This function is scripted to receive an argument held by the `checked` parameter. A variable named `showThis` is declared. An `if...else` statement is started using the `checked` parameter as the test condition. The `(checked)` expression will return either a `true` or a `false` value. If it evaluates to a `true` value, the `showThis` variable is assigned the value of the `name` form element converted to uppercase. If the `(checked)` expression returns a `false` value, the code in the `else` clause assigns the `showThis` variable the value of the `name` element converted to lowercase. After the `if...else` statement, the form element `name` is assigned the value of the `showThis` variable.

The `emailTest()` function is called with the following statement:

```
<INPUT TYPE="checkbox" onClick="(this.checked) ? ➘
emailTest(this.form) : '';">
```

The `onClick` event handler is again used to call a function. In this instance, the JavaScript conditional operator (`? :`) is used. The expression `this.checked` is the conditional expression. If `this.checked` returns a `true` value, the `emailTest()` function is invoked. If `this.checked` returns a `false` value, the code following the colon will execute. As in a previous example, an empty set of quotation marks indicates to the program that nothing should happen. When the `emailtest()` function is called, `this.form` is passed as an argument. This makes the values of the form's elements available to the function.

Following is the emailTest() function:

```
function emailTest(form)  {
    if (form.email_address.value.indexOf("@",0) < 0) {
        alert("This is not a valid e-mail address!");
    } else {
        alert("This could be a valid e-mail address");
    }
}
```

INSTRUCTOR NOTE: The String object also provides the search() method. Just as with the indexOf() method, the search() method returns the index number where the matching string first is found in the search string, or a -1 if it is not found. However, the argument to the search() method is a regular expression.

This function contains an if...else statement. The conditional expression uses the indexOf() method to determine if the value entered in the email_address form element contains the @ symbol. The conditional expression checks to see whether the value returned by the indexOf() method is less than 0. Remember that a -1 will be returned if the search text is not found in the string. The appropriate alert() method will then execute based on the Boolean value returned by the conditional expression.

The showFirst2() function is called with the following statement:

```
<INPUT TYPE="button" VALUE="first two characters of phone number"
onClick="showFirst2(this.form);">
```

This statement uses the button object's onClick event handler to invoke the showFirst2() function. Following is the showFirst2() function:

```
function showFirst2(form)  {
  var first2Chars;
  first2Chars = form.phone_number.value.substring(0, 2);
  alert(first2Chars);
}
```

In this function, a variable named first2Chars is declared. The variable is assigned the value returned by applying the substring() method to the value held in the phone_number form element. The argument for the substring() method, (0, 2), instructs the method to extract the string starting at index position 0 and ending at the character before index position 2. The string will then contain the characters in index position 0 and index position 1, the first two characters of the string. An alert() method then displays the extracted string.

The final block of code contains another example of inline scripting. Consider this block of code:

```
<INPUT TYPE="button" VALUE="fax number length"
    onClick="var strLength = document.myForm.fax_number.value.length;
    alert('That string is ' + strLength + ' characters long');
    ">
```

This code demonstrates that it is possible to script multiple lines of code that will execute via the onClick event handler. As with any JavaScript code block, each line to be executed should end with a semicolon. In this example, the first line of code declares a variable named strLength. The variable is assigned the value returned by determining the length of the fax_number form element's value. The String object's length property is used to determine this value. An alert() method then displays the strLength variable concatenated into the appropriate text.

JavaScript Regular Expressions

The JavaScript 1.2 specification added regular expressions to the JavaScript language. Regular expressions provide a mechanism whereby programmers can search for specified patterns in text. Regular expressions have long been a part of many programming environments, particularly in languages used with the UNIX operating system. The addition of regular expressions to the JavaScript language provides powerful pattern-matching capabilities for JavaScript programs.

JavaScript provides the built-in `RegExp` object to work with regular expressions. However, literal strings can use regular expressions without creating an explicit `RegExp` object. This ability follows JavaScript's approach to strings in general. You can use regular expressions without creating a formal `RegExp` object just as you can use `String` properties and methods without creating a formal `String` object.

Regular expressions use a string of characters and/or symbols to identify a pattern of text. The defined pattern is then used to search another string to determine if the pattern exists within that string. The search can be used to validate the contents of the string, or a replacement operation can be performed. As you have learned, the `indexOf()` and `lastIndexOf()` methods can be used to perform these types of operations, and are especially efficient when the search text and string match (or do not match) identically. The power of regular expressions becomes apparent, however, when you need to perform a different type of operation. For instance, suppose you need to check a string to ensure that it matches a telephone number. You cannot check for each and every literal telephone number. With regular expressions, you can easily create a telephone number pattern against which you could check any string.

INSTRUCTOR NOTE: Mastering regular expressions in JavaScript will help the developer in other programming environments. Languages such as Perl, Java, PHP, Visual Basic and VBScript all provide regular expression functionality.

The following examples demonstrate the creation of two simple regular expressions. These regular expressions characterize the same type of operation you could perform with the `indexOf()` method. The created pattern is merely a string of text you need to match. The first example uses literal text whereas the second example uses the `RegExp` object to create a pattern:

```
var myRegExp = /Juan/;

var myRegExp = new RegExp("Juan");
```

In the first example, the regular expression is delimited by two forward slashes. The first slash marks the beginning of the regular expression and the second slash marks the end of the regular expression. In the second example, a `new RegExp` constructor creates a `RegExp` object with the constructor's argument (the desired pattern) placed in quotation marks and within parentheses. Both methods of creating a regular expression are acceptable, and both result in the creation of a regular expression object. However, the first example is shorter and ultimately more efficient.

After a pattern has been created, it can be used to perform various string operations. For instance, the `RegExp` object provides the `test()` method used to test a string for a matching pattern. Consider the following example:

```
var myNames = "Paul, George, Ringo, John";
var myRegExp = /John/;
document.write(myRegExp.test(myNames));
```

The output of the preceding example would be `true` because the pattern `/John/` was found in the `myNames` string. Note that the `test()` method is invoked by a variable representing a regular expression. The string to be tested is passed as an argument to the `test()` method.

The `String` object provides several methods that can be used with plain text or regular expressions. These include the `split()`, `replace()`, `search()` and `match()` methods. The `split()` method splits a string and returns an array of substrings. The `search()` method searches a string for a string of text and returns the index position of the string, if found. The `match()` method is similar to the `search()` method, except that it returns an array of matched text instead of an index position. Consult a JavaScript reference for more information about the `split()`, `search()` and `match()` methods. The `replace()` method is demonstrated in the following example:

```
var myNames = "Paul, George, Ringo, John";
var myRegExp = /John/;
myNames = myNames.replace(myRegExp, "Johnny");
document.write(myNames);
```

The output from this example would be `Paul, George, Ringo, Johnny`. The `replace()` method receives two arguments. The first is the string to be replaced, and the second is the replacement text.

All the operations demonstrated thus far could have been performed with the `indexOf()` method, and some string-parsing code in the case of the `replace()` method example. You can use regular expressions as the preceding examples have demonstrated, but their true power becomes apparent when you create patterns that are not plain text. Consider the following example:

```
var myTeleNum = "555-123-1234";
var myRegExp = /\d{3}-?\d{3}-?\d{4}/;
document.write(myRegExp.test(myTeleNum));
```

The output from this example would be `true` because the value held by the `myTeleNum` variable matches the pattern created by `myRegExp`. The regular expression created in the preceding example appears to be quite complex, but as you dissect each portion and examine it, the pattern becomes clear.

As you have already learned, the backslash characters delimit the pattern. The first symbol in the pattern is the `\d` symbol. This symbol is referred to as a metacharacter and is used to represent any numeral between 0 and 9. Following the `\d` is `{3}`, which is a counting metacharacter used to indicate that the character immediately preceding it must occur a specified number of times. In this case, a pattern has been started that must begin with any three numerals from 0 to 9. The next character is a literal dash. The next symbol, a question mark (?), is also a counting metacharacter. This metacharacter indicates that the preceding character (the dash) must occur zero times or one time. The next group of characters, `\d{3}`, means that the pattern must then contain any three numerals from 0 to 9. Another literal dash follows with a `?` metacharacter indicating the dash must occur zero times or one time. The last group of characters, `\d{4}`, indicates the pattern must end with any four numerals from 0 to 9.

INSTRUCTOR NOTE: JavaScript also provides the **`exec()`** method to work with regular expressions. This method is called as a method of a regular expression. If needed, refer to JavaScript documentation for information about this method.

Figure 6-6 illustrates the regular expression described in the preceding paragraph.

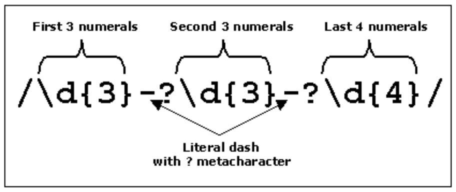


Figure 6-6: Telephone number regular expression

The telephone number regular expression could be used to validate user input. In this case, the user could enter a telephone number either in the format `555-123-1234` or in the format `5551231234`. This option is offered because the `-?` segment of the regular expression indicates that the dash can be omitted or can be present one time.

Regular expressions offer a wide variety of metacharacters that can be used to create extremely complex patterns. The use of regular expressions can enhance your Web application with powerful text search and replacement capabilities. The use of advanced regular expressions is beyond the scope of this course.

The *Array* Object

INSTRUCTOR NOTE: Refer to **Instructor Note 6-4** for information concerning multi-dimensional arrays in JavaScript.

All major programming languages use the array concept. You can think of an array as a single variable with multiple "slots" for different values. Each slot can be referenced by its index number. In JavaScript, array index numbers are zero-based, meaning the first slot in any array has a default index number of 0.

Some of the objects you have already used have arrays built into them. The `window` object, for example, has a built-in frames array that allows you to reference frames by their index number within that array. In another lesson, you will see that form fields can be referenced in a form elements array.

In addition to these intrinsic arrays, you can create your own arrays. You can declare any variable to be an array variable. You can declare that a variable will hold an array of values using the following syntax:

```
var questions = new Array();
```

This example uses the built-in language object of the `Array` as a template, and creates a new instance of the `Array` based on the existing template object.

An alternative way to declare an array is as follows:

```
var questions = new Array(10);
```

INSTRUCTOR NOTE: This section demonstrates to the students how to create `Array` objects. The students will apply this information in the next lab.

This example defines an array, and also predesignates the maximum number of slots or values it will contain. JavaScript recognizes a total of 10 elements in this array with index numbers 0 through 9. However, presizing an array provides no real benefit because a value can be assigned to any slot in an array as needed. The array's length property (discussed later in this lesson) will automatically adjust its value to accommodate the resized array.

To define individual values in the array, you list the variable name followed by square brackets and an index number within the brackets, starting from `0`:

```
var questions = new Array();
questions[0] = "What is 5 * 6?";
questions[1] = "Who played Scarlett O'Hara?";
questions[2] = "Which horse won the 1980 Irish Derby?";
```

In programming, the integer inside the square brackets is known as a subscript.

Suppose that instead of the last line of code shown in the preceding example, you accidentally entered the following:

```
questions[200] = "Which horse won the 1980 Irish Derby?";
```

In this case, you used an extremely large amount of system memory by creating 201 slots when you only needed three. In this example, `questions[2]` through `questions[199]` would return `undefined` if accessed in a script block.

JavaScript provides an alternative syntax when creating an array. This syntax is demonstrated in the following example:

```
var answers = new Array(30, "Vivien Leigh", "Tyrnavos");
```

This syntax is useful when creating arrays with relatively few elements, but can be used to create an array with any number of elements. The element's values are entered in a comma-delimited list. In the preceding example, note that the value for the integer is not placed in quotation marks whereas the string literals are. The following syntax could then be used to access the `answers` array elements:

```
document.write(answers[0]); // outputs 30
document.write(answers[1]); // outputs Vivien Leigh
document.write(answers[2]); // outputs Tyrnavos
```

JavaScript 1.2 provides yet another method for creating and populating an array. Consider the following example:

```
var answers = [30, "Vivien Leigh", "Tyrnavos"];
```

The square brackets operate like a call to the `new Array` constructor. Remember that the user agent must support JavaScript 1.2 to use the syntax demonstrated in the preceding example.

The following sections will briefly examine some commonly used `Array` object methods.

The *join()* method

The `join()` method creates a string of an array's values. The `join()` method requires a single argument: The character that will act as the separator for the element's values. Consider the following example:

```
var myArray = new Array("value 1", "value 2", "value 3");
var myArrayString = myArray.join(",");
document.write(myArrayString);
```

The output of the preceding example would be as follows:

```
value 1,value 2,value 3
```

INSTRUCTOR NOTE: Stress to students that using the `Array` object's `reverse()` method permanently alters the original array. This is true even if a new variable is assigned the result of the `reverse()` method as in:

```
var aNewArray =
myArray.reverse(
);
```

If the original array is needed later in the script, copy the array into another variable before executing the `reverse()` method on the array.

Note that the original array remains unaltered by this method. This method is especially useful when you need to send an array's values to a form field, or perhaps to a CGI program that is expecting a string instead of an array of values.

The *reverse()* method

The `reverse()` method is used to reverse the order of an array's elements. In this method, the last element in the array would then occupy the 0 index position. Consider the following example:

```
var myArray = new Array("value 1", "value 2", "value 3");
myArray.reverse();
document.write(myArray);
```

Note that the preceding example includes the `document.write(myArray);` statement. Instead of accessing individual array elements, JavaScript will output the array element values in a comma-delimited list. The output of the preceding example would be as follows:

```
value 3,value 2,value 1
```

Note that the original array is altered by this method.

The *sort()* method

The `sort()` method arranges the elements of an array in a particular order. This method can use a user-defined function to sort an array, or it can be used in a default manner. When used without a function, the `sort()` method converts the elements of the array to strings and performs an alphabetical sort of the values. The `sort()` method uses the ASCII equivalent characters for the sort operation. This feature has strong implications if your array contains numeric values and you are using the `sort()` method to arrange the array. Consider an array whose elements contain string values:

```
var myArray = new Array("my dog", "your cat", "her fish");
myArray.sort();
document.write(myArray);
```

The output of the preceding example would be as follows:

```
her fish,my dog,your cat
```

As the preceding example demonstrates, the `sort()` method performs as expected for string values. See the related appendix for information concerning the `sort()` method when used with array elements whose values are numeric.

It should be noted that when the `sort()` method is used to sort an array, the array is not copied but actually rearranged. Using the previous example, after sorting, `myArray[0]` will return the value `her fish`. This concept will be demonstrated in the next lab.

The *Array* object length property

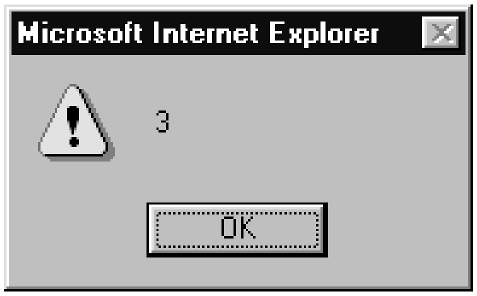
INSTRUCTOR NOTE: This section demonstrates to the students how to output the values of an `Array` object. The students will apply this information in the next lab.

Every `Array` object has a length property that returns the total number of elements in an array. Remember that an array with three elements will have index numbers of `0`, `1` and `2`.

Using the `questions` array example, suppose you typed the following:

```
alert(questions.length);
```

You would see a message as shown in Figure 6-7.



```
3
```

Figure 6-7: `questions` array has length of 3

The `length` property is especially valuable when using a loop construct to access the array's elements. The following example uses the `questions` array previously discussed:

```
var len = questions.length;

for (i = 0; i < len; i++) {
  document.write(questions[i] + "<P>");
}
```

Note that the loop counter variable `i` is placed in the square brackets following the array name. Each time through the loop, the appropriate index number held by `i` is inserted into the square brackets, thus accessing each element in the `questions` array. The output from this example would be as follows:

```
What is 5 * 6?

Who played Scarlett O'Hara?

Which horse won the 1980 Irish Derby?
```

The following lab provides an opportunity for you to use the `Array` object in a JavaScript program.

Lab 6-3: Creating an *Array* object

1. **Editor:** Open the **lab6-3.htm** file from the Lesson 6 folder of the Student_Files directory.

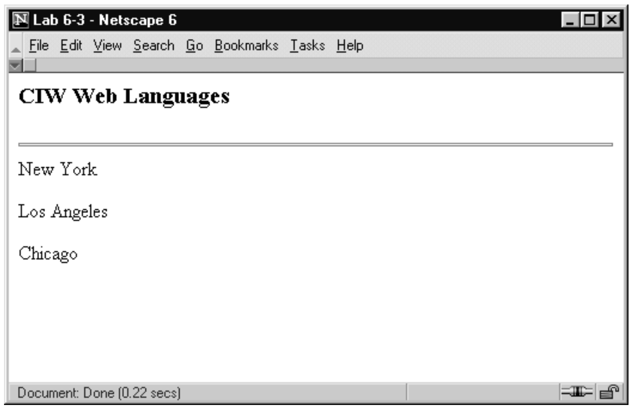
2. **Editor:** Locate the existing <SCRIPT> tag in the <HEAD> section of the document. Create an array named `citiesArray`. Add three elements to `citiesArray`: **New York**, **Los Angeles** and **Chicago** (in that order). Use any valid syntax previously discussed in this lesson to add the array elements to the array.

3. **Editor:** Locate the existing <SCRIPT> tag in the <BODY> section of the document. Create a variable name `len`. Assign the result of the `citiesArray length` property as the value for `len`.

4. **Editor:** Next, create a **`for`** loop with a loop counter variable initialized to **`0`**. The second argument in the `for` loop will use the `len` variable to determine the number of times that the loop will execute. Inside the loop, use a `document.write()` statement to output the elements of the `citiesArray` with a paragraph break between each one.

5. **Editor:** Save **lab6-3.htm**.

6. **Browser:** Open **lab6-3.htm**. Your screen should resemble Figure 6-8. If not, verify that the source code you entered is correct.



CIW Web Languages

New York

Los Angeles

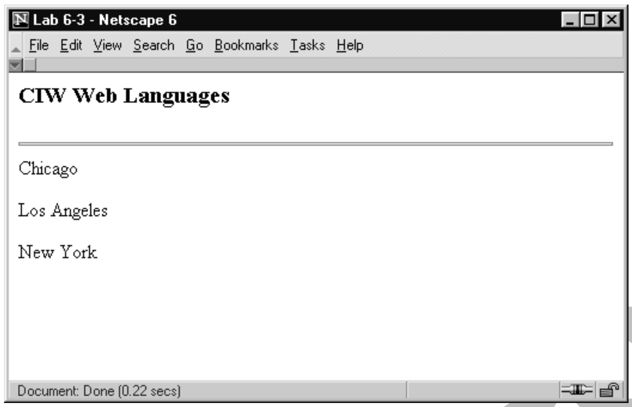
Chicago

Document: Done (0.22 secs)

Figure 6-8: Lab6-3.htm

7. **Editor:** Use the `sort()` method to sort `citiesArray`. Save **lab6-3.htm**.

8. **Browser:** Refresh **lab6-3.htm**. Your screen should resemble Figure 6-9.



CIW Web Languages

Chicago

Los Angeles

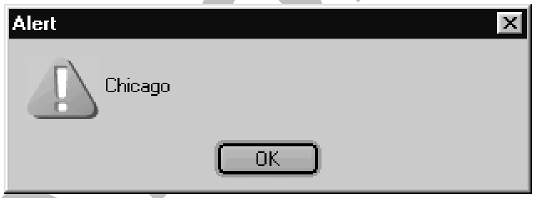
New York

Document: Done (0.22 secs)

Figure 6-9: Lab6-3.htm with sorted array

9. **Editor:** After the `for` loop, add an alert that outputs the value of `citiesArray[0]`. This will illustrate the fact that the array has been rearranged, not just temporarily copied. Save **lab6-3.htm**.

10. **Browser:** Refresh **lab6-3.htm**. You should see an alert as shown in Figure 6-10.



Chicago

Figure 6-10: Alert dialog box

In this lab, you created an `Array` object and added elements to the array. You also saw that using the `Array` object and its `length` property with a `for` statement allowed you to write a few lines of code to generate three lines of output. However, had `citiesArray` contained 1,000 elements, the same few lines of code would have generated 1,000 lines of output. Using a loop construct to access array values is a very common operation in JavaScript. The index numbers for arrays begin with `0`, and a loop counter variable is typically initialized to `0`. Therefore a `for` loop is a natural tool for accessing an array's values.

The `Array` object can be used for many operations. In fact, the `Array` object provides a powerful mechanism for creating client-side databases. You will see examples of these client-side databases later in this course.

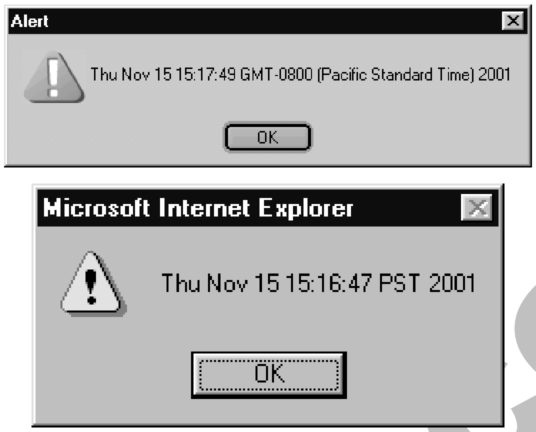
INSTRUCTOR NOTE: The actual value of the JavaScript `Date` object is the number of milliseconds since January 1, 1970 at 00:00:00.

The *Date* Object

The only way to use date and time information in JavaScript is through the `Date` object. To use any date or time information, you must create a new instance or copy of the `Date` object, as follows:

```
var myDateObject = new Date();
alert(myDateObject);
```

This statement would create an object named `myDateObject` that would, for the moment, contain the current system date and time information. Figure 6-11 shows the previous code example executed in Netscape Navigator and Microsoft Internet Explorer, respectively.



```
Thu Nov 15 15:17:49 GMT-0800 (Pacific Standard Time) 2001
```

```
Thu Nov 15 15:16:47 PST 2001
```

Figure 6-11: `Date` object information

INSTRUCTOR NOTE: This section demonstrates how to create a `Date` object. This section also demonstrates the use of a `Date` object method. The students will use this information in the next lab. They will also review code in the next lab that uses this information.

The `Date` object has built-in methods that allow you to access and change the date or time that is stored in your `Date` object. For instance, to access the number representing the day of the month, the following could be used:

```
var myDateObject = new Date();
var myDay = myDateObject.getDate();
alert(myDay);
```

This example would output an alert dialog box with the number of the day of the month.

Table 6-3 lists commonly used methods associated with the `Date` object. Each method returns (gets) or assigns (sets) the date information from a `Date` object. Several methods also return date or time information in various formats.

INSTRUCTOR NOTE: The `Date` object also provides methods such as `getUTCFullYear()` and `setUTCFullYear()`. UTC stands for Coordinated Universal Time. When calculating dates in programs that must account for the time zone of the client machine, the UTC versions of the `Date` object's methods should be used. Refer to a JavaScript reference for information concerning the UTC versions of the `Date` object's methods.

Table 6-3: Methods of `Date` object

Method	Description
`getDate()`	Returns the day number (1 through 31).
`getDay()`	Returns the day-of-week value (0 through 6; 0 is Sunday).
`getMonth()`	Returns the month number (0 through 11; 0 is January).
`getFullYear()`	Returns the year number in four-digit format (available in Netscape Navigator 4 and later and Microsoft Internet Explorer 3 and later).
`getYear()`	Returns the number of years since 1900. (Note that in Microsoft Internet Explorer 5.5, `getYear()` returns the year in four-digit format.)
`getHours()`	Returns the hour number (0 through 23; 0 is midnight).
`getMinutes()`	Returns the number of minutes (0 through 59).
`getSeconds()`	Returns the number of seconds (0 through 59).
`getTime()`	Returns the number of milliseconds that have elapsed since 00:00:00 GMT on January 1, 1970.
`setDate(value)`	Assigns the date within the month where *`value`* is 1 through 31.
`setMonth(value)`	Assigns the month number where *`value`* is 0 through 11.

Table 6-3: Methods of `Date` object (cont'd)

Method	Description
`setYear(value)`	Assigns the year number (it is safest to assign *`value`* a four-digit year).
`setHours(value)`	Assigns the hour number where *`value`* is 0 through 23.
`setMinutes(value)`	Assigns the minute number where *`value`* is 0 through 59.
`setSeconds(value)`	Assigns the seconds number where *`value`* is 0 through 59.
`setTime(value)`	Assigns the number of milliseconds that have elapsed since 00:00:00 GMT on January 1, 1970.
`toGMTString()`	Returns the date in universal format.
`toLocaleString()`	Returns the date and time in the local system's format.
`toLocaleTimeString()`	Returns the time in the local system's format (available in Netscape Navigator 6 and Microsoft Internet Explorer 5.5 only).

The following lab illustrates some ways in which date information can be accessed and used in your scripts.

Lab 6-4: Using the *Date* object

1. **Editor:** Open the **lab6-4.htm** file from the Lesson 6 Folder of the Student_Files directory.

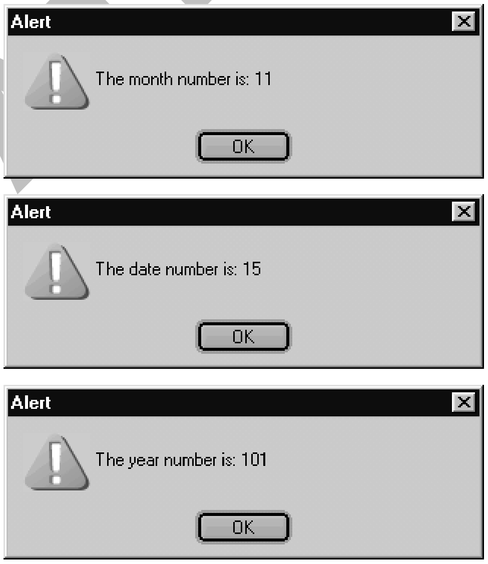
2. **Editor:** Locate the comment that reads: **`//Create Date object`**.
Create a variable named **`today`** and assign it the result of creating a `Date` object.

3. **Editor:** The following shows the code before your changes.

```
<SCRIPT LANGUAGE="JavaScript">
<!--
// Create Date object


var monthName = new Array();
monthName[0] = "January";
monthName[1] = "February";
monthName[2] = "March";
monthName[3] = "April";
monthName[4] = "May";
monthName[5] = "June";
monthName[6] = "July";
monthName[7] = "August";
monthName[8] = "September";
monthName[9] = "October";
monthName[10] = "November";
monthName[11] = "December";

var myYear = today.getYear();

if (myYear < 2000) myYear = myYear + 1900;

var myDate = today.getDate();


var dayExt = "th";

if ((myDate == 1) || (myDate == 21) || (myDate == 31)) dayExt= "st";
if ((myDate == 2) || (myDate == 22)) dayExt = "nd";
if ((myDate == 3) || (myDate == 23)) dayExt = "rd";

var extDate = myDate + dayExt;

alert("The month number is: " + (today.getMonth() + 1));
alert("The date number is: " + today.getDate());
alert("The year number is: " + today.getYear());



// -->
</SCRIPT>

</HEAD>
<BODY>
<H3>CIW Web Languages<H3>
<HR>
<H4>Today is the
<SCRIPT LANGUAGE="JavaScript">
<!--
document.write(extDate + " day of ");
document.write(monthName[today.getMonth()] + " in the year ");
document.write(myYear + ".");
// -->
</SCRIPT>
</H4>
```

4. **Editor:** Examine the rest of the source code with your instructor, then save **lab6-4.htm**.

5. **Browser:** Open **lab6-4.htm**. Three alert dialog boxes should appear indicating the month number, date number and year number, respectively, as shown in Figure 6-12.



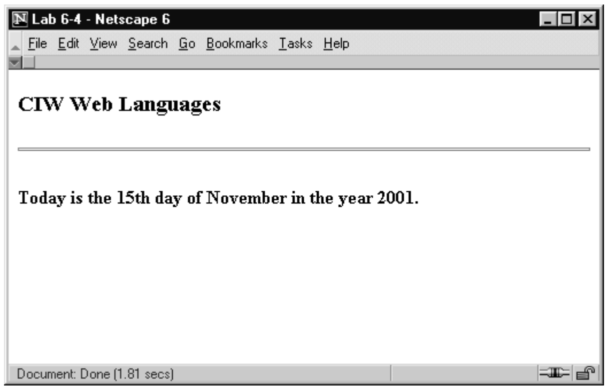
The month number is: 11

The date number is: 15

The year number is: 101

Figure 6-12: Alerts with date information

6. **Browser:** After closing the alerts, your screen should resemble Figure 6-13, except for differences in the day, month and year displayed.


Lab 6-4 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
Document: Done (1.81 secs)

CIW Web Languages

Today is the 15th day of November in the year 2001.

Figure 6-13: Date information calculated through script

In this lab, you used the `Array` object as well as the `Date` object to determine and use information about the current date. You also used a simple formula to add the appropriate suffix to the number for the day of the month (changing the cardinal number to an ordinal number, as commonly used for dates).

In the program, you created a `Date` object as the first line of code. That line declared and instantiated the variable `today`. This variable holds a reference to a new `Date` object.

Next, the `monthName` variable is declared and instantiated to hold a reference to an `Array` object. The `monthName` array elements are then populated with month names as shown in the following code:

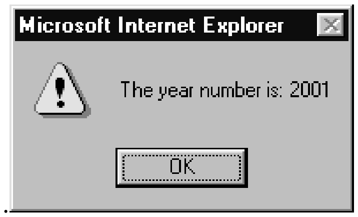
```
var monthName = new Array();
monthName[0] = "January";
monthName[1] = "February";
monthName[2] = "March";
monthName[3] = "April";
monthName[4] = "May";
monthName[5] = "June";
monthName[6] = "July";
monthName[7] = "August";
monthName[8] = "September";
monthName[9] = "October";
monthName[10] = "November";
monthName[11] = "December";
```

Note that the `monthName` array is scripted to hold 12 elements, indexed `0` through `11`.

The next two lines in the program are as follows:

```
var myYear = today.getYear();

if (myYear < 2000) myYear = myYear + 1900;
```

The variable `myYear` is declared and instantiated with the value returned by the `getYear()` method. Note that the `today` variable is used to invoke the `getYear()` method. The next line performs a check to determine the format of the information returned by the `getYear()` method. The `if` statement checks to see whether the value returned is less than 2000, and if so, adds 1900 to arrive at the proper year number. As the third alert dialog box in Figure 6-12 demonstrates, Netscape Navigator 6.2 returned `101` as the value for `today.getYear()`. If the lab6-4.htm file were opened in Microsoft Internet Explorer 6.0, that alert dialog box would appear as shown in Figure 6-14.



The year number is: 2001

Figure 6-14: Internet Explorer alert dialog box with year number

INSTRUCTOR NOTE: Refer to **Instructor Note 6-5** for another demonstration of the `Date` object.

In the preceding figure, Microsoft Internet Explorer 6.0 returns the actual year number from the `getYear()` method, and does not require any manipulation. Note that Netscape Navigator 4 (and later) and Internet Explorer 3 (and later) both support the `getFullYear()` method. This method returns the year number in four-digit format in both browsers.

The program continues with the following line of code:

```
var myDate = today.getDate();
```

This line declares and instantiates the variable `myDate` with the result of the `getDate()` method.

Next, the variable `dayExt` is declared and assigned the value `th`. This value is assigned because it is the most common extension when changing a cardinal number to an ordinal number. Next in the program are the following lines of code:

```
if ((myDate == 1) || (myDate == 21) || (myDate == 31)) dayExt➘ = "st";
if ((myDate == 2) || (myDate == 22)) dayExt = "nd";
if ((myDate == 3) || (myDate == 23)) dayExt = "rd";
```

These three `if` statements check the `myDate` variable for its value, and assign the appropriate extension to `dayExt` based on the results. Note the use of the logical OR operator (||) in the `if` statements. The next line of code is as follows:

```
var extDate = myDate + dayExt;
```

This line declares and assigns the variable `extDate` with the concatenated value of `myDate` and `dayExt`. The three `alert()` methods are next, as shown in the following code:

```
alert("The month number is: " + (today.getMonth() + 1));
alert("The date number is: " + today.getDate());
alert("The year number is: " + today.getYear());
```

Note that the `getMonth()` method returns a month number in the range of 0 to 11, with 0 representing January. To arrive at the proper month number, `1` is added to the return value of the `getMonth()` method.

The `document.write()` statements in the body of the document outputs the value held by `extDate`, the appropriate element from the `monthName` array and the value held by `myYear`. The values are concatenated into the appropriate text for output to the screen. Note the following line of code:

```
document.write(monthName[today.getMonth()] + " in the year ");
```

The interesting portion of this line is the `monthName[today.getMonth()]` portion. Again, the `getMonth()` method returns a value in the range of 0 to 11. The `monthName` array was created with elements indexed 0 to 11. As an example, for the fourth month of the year, that portion of the statement would appear as follows upon execution:

```
monthName[4]
```

No further manipulation is necessary to arrive at the proper month name.

Setting and Extracting Time Information

INSTRUCTOR NOTE: This section demonstrates how to use the `setTimeOut()` method. The students will review code in the next lab that uses this information.

Setting and extracting time information follows the same procedures as setting and extracting date information. First, you need to create a new instance of a `Date` object. Using the appropriate methods of the `Date` object as specified in Table 6-3, you can then use the time information.

In the next lab, you will review code that creates a digital clock in a form field for a page. The lab introduces a new and valuable method of the `window` object: `setTimeout()`. The `setTimeout()` method enables you to call a function based on an elapsed interval. This built-in timer object can be used to run script recursively. The syntax for the `setTimeout()` method is as follows:

```
var myTimer = setTimeout(expression, time);
```

The term ***`expression`*** refers to the command or function call you want to execute upon the elapsed interval of `time`, which is specified in milliseconds. The variable *`myTimer`* is the optional identifier you can assign to this method. The *`myTimer`* variable is needed if you intend to use the `clearTimeout()` method to stop the timer. The syntax for this method is as follows:

```
clearTimeout(myTimer);
```

If you do not need to clear the timer, you need not give it a name, and you can use the following syntax:

```
setTimeout(expression, time);
```

In the next lab, you will review code that creates a clock. The program uses the `setTimeout()` method to update the clock every second by recursively calling a function that outputs the time to a text box.

Lab 6-5: Creating an onscreen clock

1. **Editor:** Open the **lab6-5.htm** file from the Lesson 6 folder of the Student_Files directory.

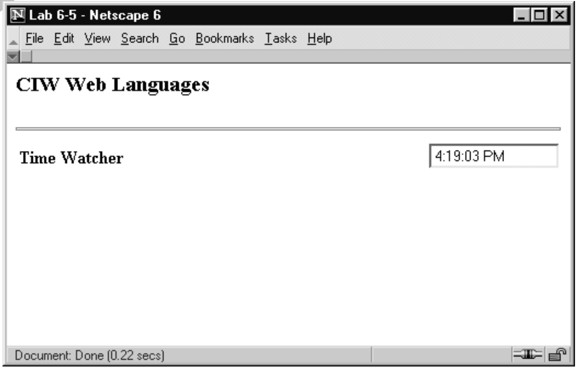
INSTRUCTOR NOTE: The code for this lab demonstrates how to assign a value to a `form` object element using JavaScript. This concept is discussed in depth elsewhere in the book.

2. **Editor:** Examine the following code with your instructor:

```
<SCRIPT LANGUAGE="JavaScript">
<!--

function runClock() {
  var timeNow = new Date();
  var hours = timeNow.getHours();
  var minutes = timeNow.getMinutes();
  var seconds = timeNow.getSeconds();
  var ampm = "";

  (seconds < 10) ? seconds = "0" + seconds : seconds;
  (minutes < 10) ? minutes = "0" + minutes : minutes;
  (hours < 12) ? ampm = "AM" : ampm = "PM";
  (hours > 12) ? hours = hours - 12 : hours;
  (hours == 0) ? hours = 12 : hours;

  var stringTime = " " + hours + ":" + minutes + ":" + ➘
                   seconds + " " + ampm;

  document.clockForm.clockBox.value = stringTime;

  setTimeout("runClock()", 1000);
}

//-->
</SCRIPT>

</HEAD>
<BODY onLoad="runClock();">
<H3>CIW Web Languages</H3>
<HR>

<TABLE WIDTH="100%">
<FORM NAME="clockForm">
<TR VALIGN="bottom">
<TD><H4>Time Watcher</H4></TD>
<TD ALIGN="right">

<INPUT TYPE="text" NAME="clockBox" size="11"  onFocus="blur();">

</TD></TR>

</FORM>
</TABLE>
```

3. **Editor:** Close the file.

4. **Browser:** Open **lab6-5.htm**. Your screen should resemble Figure 6-15.



CIW Web Languages

Time Watcher 4:19:03 PM

Document: Done (0.22 secs)

Figure 6-15: Timeclock.htm

This lab demonstrated the use of the `Date` object to create an onscreen clock. In this program, a function named `runClock()` is defined. A reference to a `Date` object is created with the `timeNow` variable. The variables `hours`, `minutes` and `seconds` are declared and assigned their values with the appropriate methods of the Date object. A variable named ampm is also declared. Next in the program are the following lines of code:

```
(seconds < 10) ? seconds = "0" + seconds : seconds;
(minutes < 10) ? minutes = "0" + minutes : minutes;
```

Both the getSeconds() and the getMinutes() methods return any value under 10 as a single digit. For instance, if the time happened to be 9:07:03, without manipulation it would be displayed as 9:7:3. The preceding lines of code check the value held in the seconds and minutes variables to determine whether either one is under 10. If so, a 0 is prepended to the value. If the value is not under 10, the current value is returned. Next is the following line of code:

```
(hours < 12) ? ampm = "AM" : ampm = "PM";
```

This line determines the appropriate value for the ampm variable. If the hours variable is less than 12, then ampm is assigned AM; otherwise ampm is assigned PM. The next line of code is as follows:

```
(hours > 12) ? hours = hours - 12 : hours;
```

This line checks the hours variable to see whether it is greater than 12. If it is, 12 is subtracted from the hours variable. If it is not greater than 12, the current value in hours is returned, so that the clock will display 1:00:00 PM instead of 13:00:00 PM, for instance. The next line of code is as follows:

```
(hours == 0) ? hours = 12 : hours;
```

This line sets the hours variable back to 12 in case the line of code before this line set it to 0. If it is not equal to 0, the current value in hours is returned, so the clock will display 12:00:00 AM instead of 0:00:00 AM. Next, the stringTime variable is built, concatenating the appropriate values with colons to create a proper time format. The next line of code is as follows:

```
document.clockForm.clockBox.value = stringTime;
```

This line assigns the value held by stringTime as the value for the form element named clockBox. The next line of code is as follows:

```
setTimeout("runClock()", 1000);
```

INSTRUCTOR NOTE: The name of the setTimeOut() method may be confusing to some new JavaScript programmers. The time-out actually refers to the amount of time before an expression executes. In JavaScript, this time-out is not a script delay. Other JavaScript statements can execute when a setTimeOut() method is called.

This line invokes the setTimeOut() method of the window object. The runClock() function name is passed as the first argument and 1000 is the second argument. The runClock() function will be called recursively every 1,000 milliseconds, or every one second. The document's <BODY> tag appears as follows:

```
<BODY onLoad="runClock();">
```

Here, the onLoad event handler is used to call the runClock() function. The setTimeOut() method ensures that the runClock() function is called every one second after this initial function call.

You may have noticed one other interesting piece of code in this lab:

```
<INPUT TYPE="text" NAME="clockBox" size="11" onFocus="blur();">
```

In the preceding line of code, the onFocus event handler is used to invoke the blur() method of a text object. This means that if the user attempts to click into the text box, the blur() method will be called. This method disallows the user from typing in the text box used to display the time information. Form elements and their methods are discussed in detail elsewhere in the book.

The *Math* Object

The `Math` object contains properties and methods that help you create advanced mathematical calculations. This object does not hold a value as you saw with the `Date` object. The `Math` object instead is a static object that contains various mathematical constants (such as the value for `PI`) as properties. You do not need to create a new instance of the `Math` object as you did with the `Date` object, nor can you reassign any of the values for the `Math` object's properties as you can with an instance of a `Date` object.

The `Math` object is most useful to programmers working in specific scientific fields of study. However, the `Math` object provides several methods that can be used to enhance various types of JavaScript programs.

The basic syntax for capturing the value generated from the `Math` object is as follows:

```
var myMathValue = Math.method(value);
```

INSTRUCTOR NOTE: All `Math` object methods excluding the `random()` method require one or more arguments when invoked. The trigonometric methods require single arguments, whereas the `min()`, `max()` and `pow()` methods require two arguments.

Table 6-4 lists commonly used methods associated with the `Math` object.

Table 6-4: Methods of `Math` object

Method	Return Value
`abs(myNum)`	Absolute value of *myNum*
`acos(myNum)`	Arc cosine of *myNum* (in radians)
`asin(myNum)`	Arc sine of *myNum* (in radians)
`atan(myNum)`	Arc tangent of *myNum* (in radians)
`ceil(myNum)`	Next integer >= *myNum*
`cos(myNum)`	Cosine of *myNum*
`exp(myNum)`	Euler's constant to the power of *myNum*
`floor(myNum)`	Next value <= *myNum*
`log(myNum)`	Natural logarithm of *myNum* (base e)
`max(myNum1, myNum2)`	Higher of *myNum1* or *myNum2*
`min(myNum1, myNum2)`	Lower of *myNum1* or *myNum2*
`pow(myNum1, myNum2)`	*myNum1* to the power of *myNum2*
`random()`	Random value from 0 to 1
`round(myNum)`	Rounds *myNum* to the nearest appropriate integer (e.g., the number 1.5 rounds to 2; the number 1.4 rounds to 1)
`sin(myNum)`	Sine of *myNum* (in radians)
`sqrt(myNum)`	Square root of *myNum*
`tan(myNum)`	Tangent of *myNum* (in radians)

Table 6-5 lists commonly used properties associated with the `Math` object.

Table 6-5: Properties of `Math` object

Property	Description	Return Value
E	Euler's constant	2.718281828459045091
LN2	Natural log of 2	0.6931471805599452862
LN10	Natural log of 10	2.302585092994045901
LOG2E	Log base-2 of E	1.442695040888963387
LOG10E	Log base-10 of E	0.4342944819032518167
PI	Pi	3.141592653589793115
SQRT1_2	Square root of 0.5	0.7071067811865475727
SQRT2	Square root of 2	1.414213562373095145

Using the *Math* object

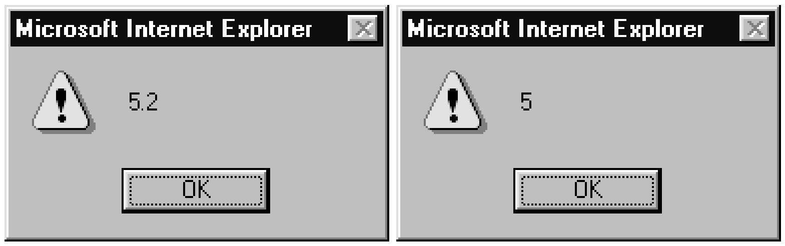
INSTRUCTOR NOTE: This section demonstrates to the students how to use the `Math` object. The students may use this information in an optional lab.

The following source code demonstrates the use of the `round()` method of the `Math` object. In this example, the number 5.2 is rounded to 5.

```
<HTML>
<HEAD>
<TITLE>Certified Internet Webmaster JavaScript Professional
</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--

var num = 5.2;
alert(num);

var newNum = Math.round(num);
alert(newNum);

// -->
</SCRIPT>
</HEAD>

<BODY>
</BODY>
</HTML>
```

INSTRUCTOR NOTE: The `Math` object's properties can be compared to other programming languages' mathematical constants. The `Math` object's methods can be compared to other languages' mathematical functions.

If this source code were executed in your browser, you would see the alert screens shown in Figure 6-16.


Microsoft Internet Explorer

5.2

Microsoft Internet Explorer

5

Figure 6-16: `Math` object alert box sequence

The optional lab for this lesson demonstrates the use of two methods of the `Math` object.

Lesson Summary

Application project

This project will challenge you to use some of what you have learned in this lesson.

This application will present the user with an HTML page containing two text entry areas. The user will input values in each, then select a button that will invoke a JavaScript function. The function will be scripted to search the second data entry for the first data entry. In other words, the program will search the second string entered to see whether it contains the first string entered. The application will require that your script detect the text entries made by the user. A template has been created that contains enough code to get you started. This template, named **lesson6AppProj.htm**, is located in the Lesson 6 Application Project folder inside the Lesson 6 folder of the Student_Files directory.

The template file is an HTML page with a `<SCRIPT>` block in the `<HEAD>` section of the document. A function named `search()` has been started for you. The data entered by the user is held in two arguments named `sub` and `string`. The `sub` argument represents the data entered in the first text box and is the string for which a search is made. The `string` argument represents the data entered in the second text box and is the string being searched.

Your task is to create code that will search for the first string within the second string. Use an appropriate method of the `String` object to determine whether the string exists within the search string. If the string is found, alert the user with the index position at which the string was found. If the string is not found, provide the user with an appropriate message.

Skills review

JavaScript provides the `String` object to work with strings in your scripts. The `String` object provides methods and properties that can be used to format, manipulate and verify the content of strings. The JavaScript 1.2 specification added regular expressions to the JavaScript language. Regular expressions provide a mechanism whereby programmers can search for specified patterns in text. JavaScript also provides the `Array` object; an array is a single variable with multiple "slots" for different values. In JavaScript, the `Date` object is used to access and manipulate date and time information in your scripts. The `Math` object contains properties and methods that help you create advanced mathematical calculations. The `Math` object contains various mathematical constants (such as the value for `PI`) as properties.

Now that you have completed this lesson, you should be able to:

- ✓ Use the `String` object to test user input.
- ✓ Identify basic regular expressions and the `RegExp` object.
- ✓ Deploy the `Array` object to create more efficient code.
- ✓ Identify uses for the `Date` and `Math` objects.

Lesson 6 Review

1. Name the JavaScript language objects discussed in this lesson.

   `String`, `Math`, `Array`, `Date` and `RegExp`.

2. What can a string consist of? What are two ways strings can exist in JavaScript?

   A string can consist of text, numbers, or any combination of characters that functions as text. Strings can exist as literals and as values of variables.

3. What is a constructor?

   A mechanism that creates an object in JavaScript.

4. Name at least two uses for which the `String` object provides methods?

   Some examples include formatting text; testing for the presence of certain characters; extracting a subset (called a substring) from a given string; finding the position of a given character or substring in a string; and retrieving the number of characters in a string.

5. When you evaluate strings with the `indexOf()` and `lastIndexOf()` methods, what return value indicates that the sought-after string was not found?

   A return value of `-1` means the string value specified in the method was not found. (Some versions of Navigator may return an empty string instead of `-1` when using the `indexOf()` method.)

6. What are regular expressions? What JavaScript object allows you to work with them?

   Regular expressions provide a mechanism whereby programmers can search for specified patterns in text. The JavaScript `RegExp` object allows you to work with regular expressions.

7. With the `RegExp` object, how do you delimit the regular expression when using literal text? How do you delimit the regular expression when creating a pattern with the `RegExp` object?

   When using literal text, delimit the regular expression with two forward slashes (//): one to mark the beginning of the regular expression and one to mark the end. When creating patterns with the `RegExp` object, use quotation marks inside parentheses after the `new RegExp` constructor.

8. What is an array? What indexing scheme does JavaScript use to make references to array elements?

   An array is a single variable with multiple "slots" for different values. Each slot can be referenced by its index number. In JavaScript, array index numbers are zero-based, meaning the first slot in any array has a default index number of 0.

9. What information does the JavaScript `Date` object allow you to access and manipulate? How can you use this information in a Web page?

   The `Date` object allows you to access and use time and date information on the client machine. You can use this information to provide or verify the current date or time, calculate projected dates for deliveries, or create an onscreen clock, for example.

10. What is the purpose of the JavaScript `Math` object? How does it differ from other JavaScript objects such as `Date`?

    The `Math` object allows you create and perform advanced mathematical calculations. Unlike other objects, the `Math` object does not hold a value; it is a static object that contains various mathematical constants as properties. Also, the `Math` object does not need to be created before its properties can be used in a script.

Lesson 6 Instructor Section

This section is a supplement containing additional tasks for students to complete in conjunction with the lesson. It also contains additional instructor notes. The instructor may use all, some or none of these additional tools, as appropriate to the specific learning environment. These elements are:

- **Additional Instructor Notes**
  Detailed instructor notes that expand the tips and suggestions presented in the instructor margin notes throughout the lesson.
- **Optional Labs**
  Computer-based labs to be completed during class or as homework.
- **Lesson Quiz**
  Multiple-choice test to assess student knowledge of lesson material.

Additional Instructor Notes

The following section contains detailed instructor notes that expand the tips and suggestions presented in the instructor margin notes throughout the lesson.

Instructor Note 6-1

Location: The String *Object*

New JavaScript programmers often forget to apply certain `String` object methods in the proper manner. For instance, the following is a common mistake:

```
var myString = "This is text.";
myString.toUpperCase();
myString.bold();
document.write(myString);
```

This code does not have the expected effect on the `myString` variable. Remind students that the return result of this type of method must be assigned to a variable as in the following:

```
myString = myString.toUpperCase();
```

Alternatively, the variable or string must be immediately manipulated as in the following:

```
document.write(myString.bold());
document.write(myString.toUpperCase());
```

This code does have the expected effect on the `myString` variable. However, for both methods to be applied at the same time, they must be stacked as in the following:

```
document.write(myString.toUpperCase().bold());
```

Instructor Note 6-2

Location: Lab 6-1: Using the String *object formatting methods*

Students may ask about the use of the `\n` special character in the code for Lab 6-1. Note that the `\n` special character has no output in the document seen in the browser window. However, the line break character does have output in the source code available through the View Source mechanism of the browser. When you create a long string of HTML to write to a dynamically created document, the string will appear as one long string in the browser's document source code. If the developer is concerned about this, the use of the `\n` special character will make the source code easier to read.

Instructor Note 6-3

Location: Additional String *Object Methods*

Students may be interested in an alternative to the `substring()` method. JavaScript1.2 introduced the `substr()` method. This method is used to extract a portion of a string, as the `substring()` method does. However, the arguments for the `substr()` method specify the beginning index position and the number of characters to extract starting at that position. Consider the following example:

```
var myString = "This is text.";
var mySubStr = myString.substr(1, 3);
var mySubString = myString.substring(1, 3)
document.write(mySubStr + " " + mySubString);
```

This code would output:

```
his hi
```

In this example, the `substr()` method starts at index position 1 and returns 3 characters. The `substring()` method starts at index position 1 and ends its string one character before index position 3.

Instructor Note 6-4

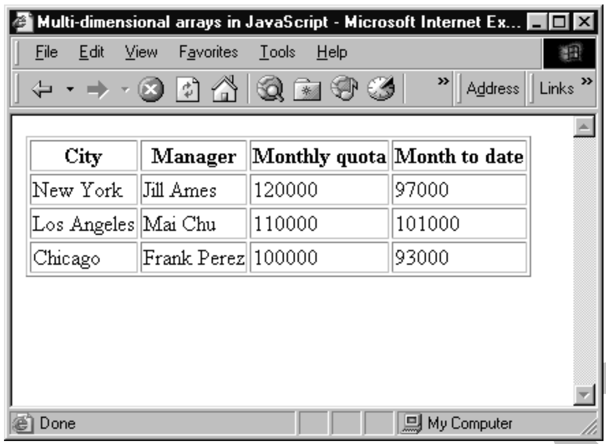
Location: The **Array** *Object*

Students experienced in other programming languages may ask about multi-dimensional arrays in JavaScript. JavaScript does not directly support multi-dimensional arrays. However, a JavaScript developer can create an array of arrays that simulates a multi-dimensional array. Consider the following example:

```
<TABLE BORDER>
<TR><TH>City<TH>Manager<TH>Monthly quota<TH>Month to date

<SCRIPT LANGUAGE="JavaScript">
<!--

var companyInfo = new Array();

companyInfo[0] = new Array("New York", "Jill Ames", 120000,97000);
companyInfo[1] = new Array("Los Angeles", "Mai Chu", 110000,101000);
companyInfo[2] = new Array("Chicago", "Frank Perez", 100000,93000);

var len1 = companyInfo.length;
var len2 = companyInfo[0].length;

for(var i = 0; i < len1; i++) {
  document.write("<TR>");
    for(j = 0; j < len2; j++) {
      document.write("<TD>" + companyInfo[i][j]);
    }
}

//-->
</SCRIPT>
</TABLE>
```

Note the assignment statement for the `len2` variable. The length of the array element `companyInfo[0]` is used as the value for the variable. Also note the syntax used to access the array of arrays element's values: `companyInfo[i][j]`.

This code would produce the output shown in Figure IN6-1.


Multi-dimensional arrays in JavaScript - Microsoft Internet Ex...
File Edit View Favorites Tools Help
Address Links
Done
My Computer

City	Manager	Monthly quota	Month to date
New York	Jill Ames	120000	97000
Los Angeles	Mai Chu	110000	101000
Chicago	Frank Perez	100000	93000

Figure IN6-1: Output of simulated multi-dimensional array

Instructor Note 6-5

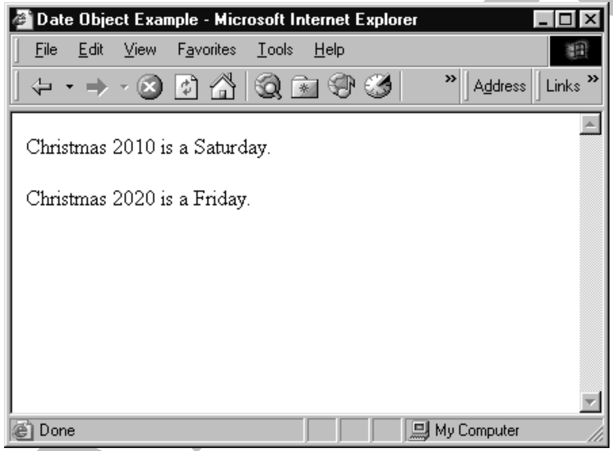
Location: The Date *Object*

This instructor note provides another example of the `Date` object. In this example, the `Date` object is used to determine the day of the week for Christmas for two different years. Examine the following example.

```
function getDayOfWeek(x) {
  switch(x) {
    case 0 :
      x = "Sunday";
      break;

    case 1 :
      x = "Monday";
      break;

    case 2 :
      x = "Tuesday";
      break;

    case 3 :
      x = "Wednesday";
      break;

    case 4 :
      x = "Thursday";
      break;

    case 5 :
      x = "Friday";
      break;

    case 6 :
      x = "Saturday";
      break;
  }
  return x;
}

var date1, dayOfweek1, dayOfWeek2

date1 = new Date("December 25, 2010");
```

```
    dayOfWeek1 = date1.getDay();
    dayOfWeek1 = getDayOfWeek(dayOfWeek1);

    date1.setYear("2020");

    dayOfWeek2 = date1.getDay();
    dayOfWeek2 = getDayOfWeek(dayOfWeek2);

    document.write("Christmas 2010 is a " + dayOfWeek1 + ".<P>");
    document.write("Christmas 2020 is a " + dayOfWeek2 + ".");
```

In this example, the `Date` object is created with a specific date. The `getDay()` method is then used to retrieve the day of the week for that date. Remember that the `getDay()` method returns an integer representing the day of the week. So a user-defined function, `getDayOfWeek()`, is called to assign the actual day name to the appropriate variable. The `setYear()` method is then used to set the year of the `date1` variable to a different year. The `getDayOfWeek()` function is then called again to retrieve the appropriate day name. The result of this example is shown in Figure IN6-2.


Date Object Example - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
Done
My Computer

Christmas 2010 is a Saturday.

Christmas 2020 is a Friday.

Figure IN6-2: Result of ***`Date`*** *object example*

Optional Lab 6-1: Using the *Math* object to generate a random quotation

In this lab, you will use the `Math` object and two of its methods. The HTML page used for this lab will display random quotations, using the `Math` object to determine which quotation will appear.

1. **Editor:** Open the **optionalLab6-1.htm** file from the Lesson 6 folder of the Student_Files directory.

2. **Editor:** You will add two lines of code to the randquotes.htm file. Locate the existing `<SCRIPT>` block in the `<BODY>` section of the document. Locate the comment that reads: **`// Use Math object`**.

3. **Editor:** Create a variable named **`num`**. Assign as its value a random integer between `0` and 9. Use the `Math` object's **`round()`** and **`random()`** methods in tandem to accomplish this task. Remember that the `random()` method generates a random number between `0` and `1`. Because the quotes array contains nine quotations, the random number generated must be multiplied by 9. Use the **`round()`** method to round that number to the nearest integer.

Hint: The `Math.random()` expression can be passed as an argument to the `Math.round()` method.

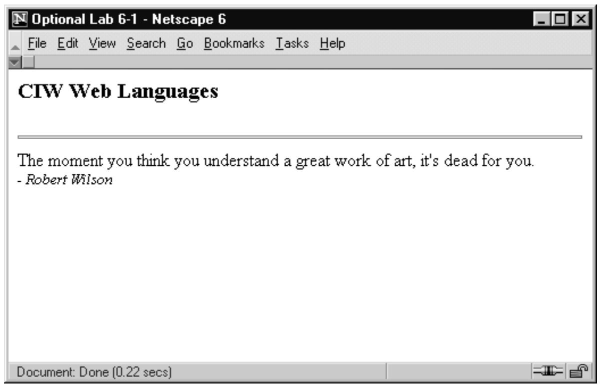
4. **Editor:** Create a **`document.write()`** statement that will output an element of the `quotes` array. Use the **`num`** variable as the subscript for the quotes array. In other words, the `num` variable should be placed inside the square brackets after the `quotes` array name. The following shows the code before your changes:

```
<SCRIPT LANGUAGE="JavaScript">
<!--

var quotes = new Array();

quotes[0] = "Every time history repeats itself the price goes➘
up.<BR><small><i>- Anonymous</i></small>";

quotes[1] = "The moment you think you understand a great work➘
of art, it's dead for you.<BR><small><i>- Robert Wilson</small></i>";

quotes[2] = "To love one person with a private love is poor ➘
and miserable; to love all is glorious.<BR><small><i>- Thomas➘
Traherne</small></i>";

quotes[3] = "Every violation of truth is not only a sort of ➘
suicide in the liar, but is a stab at the health of human➘
society.<BR><small><i>- Ralph Waldo Emerson</i></small>";

quotes[4] = "Man is to be found in reason, God in the ➘
passions.<BR><small><i>- G. C. Lichtenberg</i></small>";

quotes[5] ="Great innovations should not be forced on slender➘
majorities.<BR><small><i>- Thomas Jefferson</i></small>";

quotes[6] = "In this world nothing can be said to be certain,➘
except death and taxes.<BR><small><i>- Benjamin ➘
Franklin</i></small>";

quotes[7] = "Nine-tenths of wisdom consists in being wise in ➘
time.<BR><small><i>- Theodore Roosevelt</i></small>";

quotes[8] = "We have no more right to consume happiness ➘
without producing it than to consume wealth without producing➘
it.<BR><small><i>- George Bernard Shaw</i></small>";

quotes[9] = "So little done, so much to do.<BR><small><i>- ➘
Cecil Rhodes</i></small>";


// Use Math object


// -->
</SCRIPT>
```

5. **Editor:** Save **optionalLab6-1.htm**.

6. **Browser:** Open **optionalLab6-1.htm**. Your screen should resemble

Figure OL6-1. If it does not, verify that the source code you entered is correct.


Optional Lab 6-1 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
Document: Done (0.22 secs)

CIW Web Languages

The moment you think you understand a great work of art, it's dead for you.
- Robert Wilson

Figure OL6-1: OptionalLab6-1.htm

7. **Browser:** Reload the file several times. You will see different quotations. Because the generator is truly random, you may see the same quotation more than once, even subsequently. If you reload enough times, you will eventually see all the quotations.

In this lab, you used the `Math` object's `random()` method to generate a random number between zero and one. You multiplied that random number by nine, then used the `round()` method to create an integer used as the subscript for the `quotes` array. You then used a `document.write()` statement to output a random element from the array.

Lesson 6 Quiz

1. Which JavaScript language object allows you to access and manipulate time information in your scripts?

   a. *The `Date` object*
   b. The `Math` object
   c. The `Array` object
   d. The `String` object

2. Which JavaScript language object is a static object that does not hold a value but contains various constants as properties?

   a. The `RegExp` object
   b. The `Date` object
   c. The `Array` object
   d. *The `Math` object*

3. Which JavaScript language object allows you search for specified patterns in text??

   a. The `Math` object
   b. The `Array` object
   c. *The `RegExp` object*
   d. The `String` object

4. Which JavaScript language object allows you to give a single variable multiple "slots," each with a different value, which can be referenced by index numbers?

   a. *The `Array` object*
   b. The `Math` object
   c. The `RegExp` object
   d. The `Date` object

5. Which JavaScript language object offers predefined methods for formatting text?

   a. The `Math` object
   b. The `Array` object
   c. The `RegExp` object
   d. *The `String` object*

6. Which JavaScript language object can be used to generate random quotations from a predefined array of quotes?

   a. The `Array` object
   b. The `String` object
   c. *The `Math` object*
   d. The `RegExp` object

7. Which `String` object method can be used to test user input?

   a. The `toLowCase()` method
   b. The `test()` method
   c. *The `indexOf()` method*
   d. The `link()` method

8. Which `RegExp` object method can return the index position of a sought-after string?

   a. *The `search()` method*
   b. The `test()` method
   c. The `indexOf()` method
   d. The `match()` method

9. Which `Array` object method can be used to create a string of an array's values?

   a. The `lastIndexOf()` method
   b. The `reverse()` method
   c. The `sort()` method
   d. *The `join()` method*

10. Which `Date` object method can be used to return the day of the week as a numeric value (0 through 6, where 0 is Sunday)?

    a. The `setDate()` method
    b. *The `getDay()` method*
    c. The `getDate()` method
    d. The `setDay()` method

11. Consider the following JavaScript statements:

```
var myStr = "Caught In The Wind";
document.write(myStr.charAt(0));
document.write(myStr.charAt(7));
document.write(myStr.charAt(14));
```

    What is the output of these statements?

    `CIW`

12. Consider the following JavaScript statements:

```
var myStr = "joseB@company.com";
document.write(myStr.indexOf("@", 0) + "<BR>");
document.write(myStr.lastIndexOf("com") + "<BR>");
document.write(myStr.indexOf("b@", 0));
```

What is the output of these statements?

```
5
14
-1
```

13. Consider the following `searchString()` function:

```
function searchString(oneChar, testString) {
  var returnValue = false
  for (var i = 1; i <= testString.length; i++) {
    if (testString.substring(i, i + 1) == oneChar) {
      returnValue = true;
      break;
    }
  }
  return returnValue;
}
```

Describe the functionality of this `searchString()` function.

The `searchString()` function is designed to search a string for a particular character. The search character is received by the `oneChar` parameter, and the string to search is received by the `testString` parameter. The `returnValue` variable is declared and initialized to `false`. A `for` loop is defined, using the `length` property of the `testString` parameter as the middle condition for the loop. Inside the loop, an `if` statement is used to determine if a match has been found. The `substring()` method is used to perform this test. The `substring()` method's arguments are the loop counter variable `i`, and `i + 1`. This has the effect of examining one character of the `testString` parameter each time through the loop. If a match is found, the `returnValue` variable is set to `true`, and a `break` statement ends the loop. After the string is examined, the value of the `returnValue` variable is returned to the statement that called the `searchString()` function.

14. Consider the following array:

```
var myArray = new Array("red, "white", "blue);
```

Write code that outputs the values from this array on separate lines of an HTML page.

One way to write this code is as follows:

```
document.write(myArray[0] + "<BR>");
document.write(myArray[1] + "<BR>");
document.write(myArray[2] + "<BR>");
```

A better way to write this code is as follows:

```
for (var i = 0; i < myArray.length; i++) {
  document.write(myArray[i] + "<BR>");
}
```

15. Consider the following JavaScript statements:

```
var myArray = new Array("red", "white", "blue");
document.write(myArray);
```

What is the output of these statements?

red,white,blue

16. Consider the following JavaScript statements:

```
var myArray = new Array("red", 10, "white", 30, "blue", 5);
document.write(myArray.sort());
```

What is the output of these statements?

10,30,5,blue,red,white

17. Write a JavaScript block of code that uses the `Date` object to send the user to an HTML page titled specifically for the day of the week.

One way to write this code is as follows:

```
var myDate = new Date();
var myURL = "day" + myDate.getDay() + ".html";
location.href = myURL;
```

18. Write a JavaScript block of code that returns the current hour and displays it to the user.

One way to write this code is as follows:

```
var myDate = new Date();
var myHour = myDate.getHours();
document.write(myHour + "<BR>");
```

A better way to write this code is as follows:

```
var myDate = new Date();
var myHour = myDate.getHours();
if (myHour > 12) myHour = myHour - 12;
if (myHour == 0) myHour = 12;
document.write(myHour + "<BR>")
```

19. Write a JavaScript function that tests two numbers for the higher and outputs that number to the user.

One way to write this code is as follows:

```
function numTest(x, y) {
  document.write(Math.max(x, y));
}
```

Lesson 7: Developing Interactive Forms with JavaScript

Objectives

By the end of this lesson, you will be able to:

- Identify and use form controls.
- Refer to `form` objects.
- Define the `form` object.
- Use the `button` object.
- Use the `checkbox` object.
- Evaluate text in the `text` and `textarea` objects.
- Process `radio` object options.
- Capture choices from a select list.
- Conduct form validation.

Pre-Assessment Questions

1. Which of the following properties returns `true` if a `CHECKED` attribute is present in the HTML definition of a `checkbox` object?

   a. `checked`
   b. `value`
   c. *`defaultChecked`*
   d. `defaultValue`

2. Which of the following properties returns an integer value representing the number of `radio` objects in a radio button group?

   a. *`length`*
   b. `count`
   c. `number`
   d. `total`

3. Using the following `<SELECT>` tag definition, what is the output of the JavaScript statements when the `button` object is selected?

```
<FORM>
<SELECT NAME="languages" SIZE="5" MULTIPLE>
  <OPTION>Java
  <OPTION>JavaScript
  <OPTION>Visual Basic
  <OPTION>VBScript
  <OPTION>Perl
  <OPTION>C++
  <OPTION>Fortran
  <OPTION>Pascal
</SELECT>
<P>
<INPUT TYPE="button" VALUE="Show Selection(s)"
onClick="showSelect(this.form);">
</FORM>

<SCRIPT LANGUAGE="JavaScript">
function showSelect(form) {
  var list = "";
  var len = form.languages.length;

  for (var i = 0; i < len; i++) {
    if (form.languages.options[i].selected) {
      list += form.languages.options[i].text + "<P>";
    }
  }
}
</SCRIPT>
```

The `showSelect()` function has no output. Even though the `list` variable is populated with the selections from the `select` object, no mechanism is used to output the information. Possible methods for outputting the data include a `document.write()` statement or an `alert()` statement.

Interactive Forms

Perhaps the most common need from a Webmaster's perspective is the ability to retrieve and verify data from the user through an HTML form. Historically, the majority of scripting in HTML pages has focused on forms and form elements. As the capabilities of scripting languages and browsers grow, the primary user interface in Web pages is still the form. The Web form has proved to be a reliable, familiar and easy-to-use mechanism. Therefore, you must become proficient with the methods and techniques involved in scripting HTML form elements.

This lesson will familiarize you with the objects, methods and properties available as you create JavaScript programs that interact with forms. This lesson will conclude with examples of using JavaScript to verify user input, one of the most important services provided by JavaScript.

Overview of Form Elements

INSTRUCTOR NOTE: Each of these HTML form elements correlates to a JavaScript object of the same name. Most of these objects are created using the HTML `<INPUT>` tag with the `TYPE` attribute determining the element's appearance. Only the `select` and `textarea` objects are created with their own HTML tags.

Before you start working with JavaScript and forms, consider the available HTML form elements listed in Table 7-1. You may also want to re-examine the JavaScript object model illustrated elsewhere in the book to review where these form elements reside in the JavaScript object hierarchy.

Table 7-1: Form elements

Form Element	Description
button	A scriptable button other than a Submit or Reset button
checkbox	A check box used to submit a yes/no or true/false option
hidden	A hidden field containing a name/value pair that is submitted unseen and unchanged by the user
password	A single-line text box that displays input as asterisks (***) to mask data entry
radio	An option button in a set of mutually exclusive option buttons
reset	A Reset button used to restore default values in form elements
select	A selection list from which a single choice or multiple choices can be made
submit	A Submit button used to send the contents of a form to a server
text	A single-line text field used for data entry
textarea	A scrolling text box used for data entry

An important requirement when using form elements is to enclose the form components within a pair of opening and closing `<FORM>` tags. This requirement is obvious when using form fields to submit data to a server, but is also necessary when using form elements to invoke JavaScript functions. In fact, when you use older versions of Netscape Navigator, form elements will not appear on the page unless they are enclosed in `<FORM>...</FORM>` tags. Newer versions of Netscape Navigator display form elements if no `<FORM>` tag is present, but JavaScript cannot interact with the form elements because the required container is not present. To avoid problems, always use `<FORM>` tags to enclose form elements.

In addition, a problem exists with the way some versions of Netscape Navigator 3.0 handle forms not delivered to you from a server. The problem is that the form fields are not displayed correctly unless you reopen the page from disk when you add or change the fields. You may experience this problem in this lesson's labs if you are using Navigator 3.0.

Referring to a Form Element

INSTRUCTOR NOTE: Referencing form elements using the `elements` array is useful when scripts need to be kept generic. However, for maintainability and to make scripts more readable, referencing form elements by name is preferred.

You can refer to a form element in two ways: by its name or by its index number in the `form` object's `elements` array. Consider the following example:

```
<FORM NAME="myForm">
<INPUT TYPE="text" NAME="firstName">
</FORM>
```

In this example, the name of the form is `myForm` and the name of the text box is `firstName`. To reference the value entered in the text box, you could use either of the following two types of syntax:

```
document.forms[0].elements[0].value;
```

```
document.myForm.firstName.value;
```

In the first example, you are using the `forms` and `elements` arrays to access the first form defined in the HTML document and the first form element defined in that form. In the second example, you are referring to the form element through both the form name and the element name. Both must start with `document`, as the `form` object is subordinate to the `document` object in the JavaScript object hierarchy. You can also use a combination of the two styles, as follows:

```
document.forms[0].firstName.value;
```

The next section will introduce the `form` object. You will then examine the various form objects available to your JavaScript programs.

The *form* Object

INSTRUCTOR NOTE: If a form is being used solely for JavaScript interaction (for example, the form will not be submitted to a server for processing), the `METHOD`, `ACTION` and `ENCTYPE` attributes do not need to be present in the <FORM> tag definition.

The `form` object represents an HTML form in JavaScript. This object is available when `<FORM>` tags are present in the HTML document. Each form on a page is represented by a separate instance of the `form` object. The `form` object is used to access all properties of the form, most importantly the elements that compose the form. The generic syntax for a form is as follows:

```
<FORM NAME="formName" METHOD="submitMethod" ACTION="URL"
ENCTYPE="encodingType">
<!-- Define form elements -->
</FORM>
```

Table 7-2 describes the `form` object's commonly used properties, methods and event handlers.

Table 7-2: Features of `form` object

Property	Method	Event Handler
`action` Specifies or returns a string value specifying the URL to which the form data is submitted	`reset()` Resets form elements to their default values	`OnReset` Specifies the JavaScript code to execute when the form is reset
`elements` Returns an array of objects containing each form element in the order that the elements appear in the form	`submit()` Submits the form to a server	`onSubmit` Specifies the JavaScript code to execute when the form is submitted
`encoding` Specifies or returns a string containing the MIME encoding of the form (specified in the `ENCTYPE` attribute)		
`length` Returns the number of elements in the form		
`method` Specifies or returns a string value containing the submission method of form data to a server		
`name` Specifies or returns the name of the form as defined in the `<FORM>` tag		
`target` Specifies or returns a string value containing the name of the window to which responses to form submissions are directed		

INSTRUCTOR NOTE: This section demonstrates how to use the `elements` property of the `form` object to access form element names and values. The students may apply this information in **Optional Lab 7-1: Conducting form validation.**

The `elements` property is an important property of the `form` object. This property exposes all form elements as an array. The following example uses the `elements` property in a `for` loop to output the names and values of a form's fields.

```
function outputForm() {
var len = document.forms[0].length;
var output = "";

  for(var i = 0; i < len; i++) {
    output += document.forms[0].elements[i].name + " = " +
                 document.forms[0].elements[i].value + "\n";
  }
  alert(output);
}
```

The `form` object's `length` property is used to determine how many times the `for` loop will execute. Within the loop, the loop counter variable is used as the subscript for the `elements` array. As the code demonstrates, each form element has a `name` and a `value` property. These properties are used each run through the loop to return the appropriate information for each form element.

The *button* Object

The `button` object provides a basic push-button type of user interface element on an HTML page. Whereas Submit or Reset buttons operate in a predefined manner, the `button` object depends on scripting for its functionality. Its main event handler is `onClick`. Generic `button` syntax is as follows:

```
<INPUT TYPE="button" VALUE="Text on Button" onClick="script to execute">
```

You have seen many examples of the `button` object throughout this course. Buttons provide dependable, recognizable user interfaces, and are available on virtually every platform.

Table 7-3 describes the commonly used properties, methods and event handlers related to the `button` object.

Table 7-3: Features of ***button*** *object*

Property	Method	Event Handler
`form` Returns a reference to the `form` object containing the element	`click()` Simulates a user click on the button	`onClick` Specifies the JavaScript code to execute when the user clicks the button, or when the `click()` method is invoked
`name` Specifies or returns a string value of the name of the object as determined by the `NAME` attribute		
`value` Specifies or returns a string value of the text displayed on the button as determined by the `VALUE` attribute		

INSTRUCTOR NOTE:
A common mistake made by new JavaScript programmers is to define all selectable buttons as `TYPE="submit"` or `TYPE="reset"` in the HTML for the form. Emphasize that the `submit` object will submit a form. If submitting a form is not the intention, the button must be created as `TYPE="button"` in the HTML code. Later in this section, students will learn how to submit forms without using a `submit` object.

As you learned previously, you can call JavaScript functions from `button` objects, as in the following example:

```
<INPUT TYPE="button" Value="Help here"
onClick="getHelp();">
```

This statement would invoke a function named `getHelp()` that is defined elsewhere in the source code.

The `value` property specifies the text that will display on the button. For example, to change the text that appears on a button, you could use the following statement in your script:

```
document.formName.buttonName.value = "new value";
```

To assign the text that appears on a button to a variable, you could use its `value` property in your script as follows:

```
var buttonValue = document.formName.buttonName.value;
```

The button object also has a form property that returns a reference to the form object containing the button. The following code defines a form, a button and an onClick event handler with some associated code:

```
<FORM NAME="myForm">
<INPUT TYPE="button" NAME="myButton" VALUE="Return form name"
onClick="alert(this.form.name);">
</FORM>
```

When the button in this example is clicked, an alert dialog box with the text myForm appears. Recall that the keyword this is used to reference the object that uses it. Note that all form elements defined by <INPUT TYPE=...> tags have a form property that will return a reference to the form object containing the form element. This property provides one way to determine which form contains a form element when using multiple forms in a script. The form property is also valuable when used with event handlers to refer to another element in the current form.

A valuable use of the button object is to call a form validation function. After the form is validated by JavaScript code, the form can be submitted programmatically. Following is an example of this concept.

```
...
function validateForm() {
//form validating code

document.forms[0].submit();
}

...

<FORM NAME="myForm" METHOD="post" ACTION="http://ss1.prosofttraining.com/cgi-
bin/process.pl">
<INPUT TYPE="text" NAME="name"><BR>
<INPUT TYPE="button" VALUE="Submit" onClick="validateForm();">
</FORM>
```

When the button object is clicked, the validateForm() function executes. If the user's input satisfies the form-validating code, the form object's submit() method is invoked. This action will submit the form as if a normal Submit button were clicked.

The *checkbox* Object

A checkbox is an input object in the shape of a small square (called a check box) that can be selected, or "checked," on or off. The syntax for a checkbox object and accompanying text is as follows:

```
<INPUT TYPE="checkbox" NAME="checkboxName"
VALUE="ReturnValueIfChecked">caption text
```

For example, consider the following check box on a form:

```
<INPUT TYPE="checkbox" VALUE="Likes Swimming" NAME="swimChkBox">Do you like to
swim?
```

If a user checked this check box, the element would return the name/value pair "swimChkBox = Likes Swimming" in a form submission.

The main property associated with a checkbox object is checked. The checked property returns a Boolean value of true or false. If the box is checked, the return value is true. If the box is not checked, the checked property returns the Boolean value of false.

The main event handler associated with the `checkbox` object is `onClick`.

The following example thanks the user if the "I enjoyed this site" check box is checked:

```
<FORM NAME="responseForm">

<INPUT TYPE="checkbox" NAME="likedIt"

onClick="if (this.checked) {
alert('Thanks - glad you liked it!');
}">

Did you like this site?

</FORM>
```

Tech Note

Typically, using a check box as an action button is not a good user interface design technique. Users generally expect check boxes to present them with a choice that is either accepted or not. The previous example uses a check box as an action button. Note that this use is for demonstration purposes only, and is not a design recommendation.

INSTRUCTOR NOTE: See **Instructor Note 7-1** for important information concerning the **checked** property of the **checkbox** object.

Note in the preceding example that an `if` clause is included within the `<INPUT>` tag, with the keyword `this` used to reference the `checkbox` object. To reference a `checkbox` object in a script at a time later than it is clicked (such as when a Submit or other type of button is selected), you must refer to the object's `checked` property using the formal syntax previously discussed. For instance, you can test whether the box was checked with the following statement:

```
document.forms[0].elements[0].checked;
```

INSTRUCTOR NOTE: This section demonstrates to the students how to detect if a **checkbox** object is selected. The students will apply this information in the next lab.

Note the use of the `forms` array and the `elements` array. In this example, the referenced form is the first one defined on the HTML page. Similarly, the referenced form element is the first one defined inside the form.

You could also refer to the form and field by name, as follows:

```
document.responseForm.likedIt.checked;
```

Table 7-4 describes the commonly used properties, methods and event handlers related to the `checkbox` object.

Table 7-4: Features of ***checkbox*** *object*

Property	**Method**	**Event Handler**
`checked` Specifies or returns a Boolean value indicating `true` if the box is checked or `false` if the box is not checked	`click()` Simulates a user click on the check box	`onClick` Specifies the JavaScript code to execute when the user clicks the check box, or when the `click()` method is invoked
`defaultChecked` Returns a Boolean value indicating `true` if the CHECKED attribute was used in the `<INPUT>` tag for the `checkbox` object		

*Table 7-4: Features of **checkbox** object (cont'd)*

Property	Method	Event Handler
`form` Returns a reference to the `form` object containing the element		
`name` Specifies or returns a string value of the name of the object as determined by the `NAME` attribute		
`value` Specifies or returns a string value determined by the `VALUE` attribute		

The *text* and *textarea* Objects

INSTRUCTOR NOTE: Remind students that the return value of a **`text`** object is always a string data type. If a script is expecting a numeric value, use the **`parseInt()`** or **`parseFloat()`** functions to convert to the proper type.

To JavaScript, the `text` and `textarea` objects are very similar. To the user, of course, they are very different. A `text` object displays a single line of text, whereas a `textarea` object can display multiple, scrolling lines of text. The `text` object will be discussed here, but understand that the same properties, methods and event handlers are available to both.

The `text` object displays a text box that allows the user to enter an alphanumeric value. The generic syntax for the `text` object is as follows:

```
<INPUT TYPE="text" NAME="boxName"
VALUE="defaultValue" SIZE="size">
```

The `text` object, like the `button` and `checkbox` form objects, can include inline scripting that is called by an event. For example, if the user tabs from a text box, you may want to instantly check whether the user left the field blank. The following example demonstrates three functions: the use of the keyword `this` to refer to the current object; inline scripting that uses an `if` statement; and the invocation of the `focus()` method to return the user's cursor to the text box if the user left it blank.

```
<INPUT TYPE="text"
    NAME="userName"
    onBlur="
    if (this.value == '') {
      alert('Please do not leave the name field blank.');
      this.focus();
    }
   else {
      alert('Thank you, ' + this.value + '!');
    }
">
```

Tech Note

In some applications, immediately testing the user's input as shown in the preceding example may be a good design technique. However, in some programs, it may be preferable to check all user input at one time, usually upon submission. Test your form validation scripts carefully to ensure that you use the technique that is most appropriate for the application.

INSTRUCTOR NOTE: The `text` object's `change` event and the **onChange** event handler may confuse new JavaScript programmers. Remind students that code associated with the **onChange** event handler will not execute until the `text` element loses focus.

Table 7-5 describes the commonly used properties, methods and event handlers related to the `text` and `textarea` objects.

Table 7-5: Features of ***`text`*** *and* ***`textarea`*** *objects*

Property	Method	Event Handler
`defaultValue` Returns a string value indicating the value as specified in the <INPUT> tag with the VALUE attribute	`blur()` Moves the cursor away from the object, causing it to lose focus or blur	`onBlur` Specifies the JavaScript code to execute when the user moves the cursor away from the object
`form` Returns a reference to the `form` object containing the element		`onChange` Specifies the JavaScript code to execute when the object loses focus after a user enters text
`name` Specifies or returns a string value of the name of the object as specified in the <INPUT> tag with the NAME attribute	`focus()` Moves the cursor to the object, giving it focus	`onFocus` Specifies the JavaScript code to execute when the user moves the cursor to the object
`value` Specifies or returns a string value of the current text box contents	`select()` Highlights (selects) the text in the object	`onSelect` Specifies the JavaScript code to execute when the user highlights (selects) the text in the object *Note: JavaScript has a known problem with this event handler. As of the time of this writing, it does not work properly in all browsers.*

Determining the value that a user has entered in a text box is a common operation in a JavaScript program. The following example demonstrates a small program that gathers the input from a text box and displays it in an alert dialog box. The form definition is as follows:

```
<FORM NAME="textForm">
<INPUT TYPE="text" name="text1">
<P>
<INPUT TYPE="text" name="text2">
<BR>
<INPUT TYPE="button" VALUE="Check Text" NAME="myButton"
onClick="checkText();">
</FORM>
```

INSTRUCTOR NOTE: This section demonstrates how to detect the value entered in a `text` object. The students will apply this information in the next lab.

Note that the `checkText()` function is called using the `onClick` event handler of the `button` object. The `checkText()` function is as follows:

```
function checkText() {
  var myForm = document.textForm;
  var firstValue = myForm.text1.value;
  var secondValue = myForm.text2.value;
  alert("First text box : " + firstValue + "\n" + ➘
        "Second text box : " + secondValue);
}
```

The `myForm` variable is assigned the value `document.textForm`, allowing the programmer to easily refer to the form without having to type `document.textForm` repeatedly. Two variables, `firstValue` and `secondValue`, are assigned the appropriate values using the value property of the text object. The variables are then concatenated into the alert() method's argument.

INSTRUCTOR NOTE: This section demonstrates how to pass form element values when invoking a form validation function. The students will use this information in several labs throughout this lesson.

JavaScript provides another technique for accessing form element values. Examine the following form:

```
<FORM NAME="textForm">
<INPUT TYPE="text" name="text1">
<P>
<INPUT TYPE="text" name="text2">
<BR>
<INPUT TYPE="button" VALUE="Check Text" NAME="myButton"
onClick="checkText(this.form);">
</FORM>
```

Note the argument passed to the checkText() function. The argument this.form sends all name/value pairs for all form elements to the function. Examine the revised checkText() function:

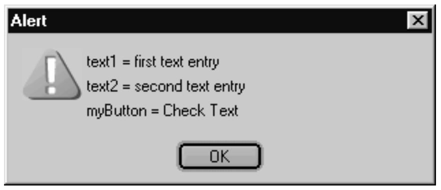
```
function checkText(form) {
var firstValue = form.text1.value;
var secondValue = form.text2.value;
alert("First text box : " + firstValue + "\n" + ➘
      "Second text box : " + secondValue);
}
```

The parameter form receives all name/value pairs from the form. The simpler syntax form.text1.value can now be used to access that element's value. In many programs, a loop is used to access form element values, especially in data validation scripts. Examine the following function:

```
function checkForm(form) {
  var values = "";
  for (var i = 0; i < form.length; i++) {
    values += form.elements[i].name + " = ";
    values += form.elements[i].value + "\n";
  }
  alert(values);
}
```

The checkForm() function uses the same mechanism previously discussed to receive the name/value pairs from the form. In the checkForm()function, a variable named values is declared and initialized to an empty string. A for loop is started, using the expression form.length to determine how many times the loop will execute. Recall that the length property of the form object returns the number of elements in the form. Inside the loop, the values variable is assigned the name and value of each form element followed by a new line character. The elements array is used, along with the name and value properties of the text object. Note that the loop counter variable is placed inside the square brackets after the elements array name. The values variable is then displayed in an alert dialog box.

Although the `checkForm()` function did not perform any data validation, the mechanism used to access the values of the form elements could be used in a form validation routine. Upon execution, the `checkForm()` function used in the previous example would result in an alert dialog box similar to the one shown in Figure 7-1.



```
text1 = first text entry
text2 = second text entry
myButton = Check Text
```

Figure 7-1: Alert with form name/value pairs

INSTRUCTOR NOTE: If needed, the `text` object provides a `defaultValue` property used to return the string defined by the VALUE attribute of the <INPUT> tag.

As shown, the contents of the form are sent to the `checkForm()` function and are received in the `form` parameter. Note that this also includes the `button` object defined in the form.

The following lab provides an opportunity to work with form elements in your JavaScript code.

Lab 7-1: Using a text box, a check box and a button

1. **Editor:** Open the **lab7-1.htm** file from the Lesson_7 folder of the Student_Files directory.

2. **Editor:** An HTML form has been created for you. Note that the `text` object in the form is named `myText`. Note also that a `button` object is scripted to call a function named `checkText()`. As previously discussed, the argument `this.form` passes the form's name/value pairs to the function.

3. **Editor:** Locate the existing <SCRIPT> tag in the document's <HEAD> section. Locate the **`checkText()`** function that has been started for you. You will add two lines of code to this function. Locate the comment that reads as follows:

   ```
   // Check myText element value
   ```

   Create a variable named `myValue` and assign as its value the text entered in the `myText` form element. Then create an alert dialog box that reflects this information back to the user.

4. **Editor:** Save **lab7-1.htm**.

5. **Browser:** Open **lab7-1.htm**. Your screen should resemble Figure 7-2.

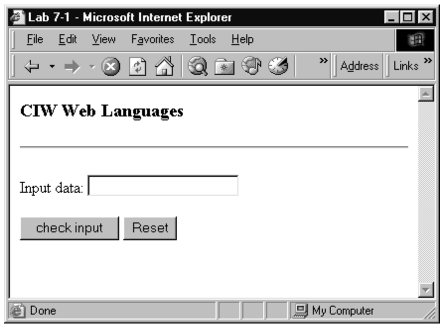

Lab 7-1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
CIW Web Languages
Input data:
check input
Reset
Done
My Computer

Figure 7-2: Lab7-1.htm

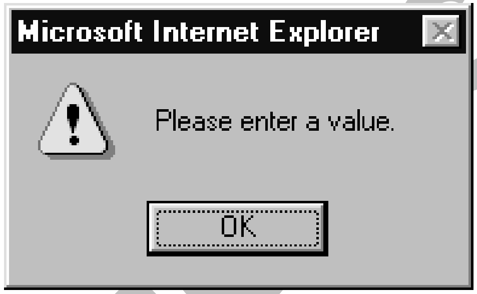
6. **Browser:** Enter data in the text field. Click the **check input** button. You should see an alert dialog box containing the text entered in the text field. If you do not, verify that the source code you entered is correct.

INSTRUCTOR NOTE: Refer to **Instructor Note 7-2** for important information concerning the `text` object's `focus()` and `select()` methods.

7. **Editor:** Save **lab7-1.htm** as **lab7-1.1.htm**. You will now add code to lab7-1.1.htm that performs simple form validation. Following the `myValue` variable declaration, create an `if` statement that checks `myValue` for a value. If `myValue` is empty, create an alert dialog box that asks the user to enter data. Then use the `focus()` method to place the user's cursor in the text box. As the next line of code, use the `return` keyword to end the function. Remember to properly close the `if` statement. As in lab7-1.htm, if user input is present, it should appear in an alert dialog box.

8. **Editor:** Save **lab7-1.1.htm**.

9. **Browser:** Open **lab7-1.1.htm**. Your screen should resemble Figure 7-2. Without entering data, click the **check input** button. You should see an alert similar to Figure 7-3. If you do not, verify that the source code you entered is correct.



Please enter a value.

Figure 7-3: Alert dialog box

10. **Editor:** Open the **lab7-1.2.htm** file from the Lesson_7 folder of the Student_Files directory. This file contains an HTML form with a `checkbox` object named `myCheckBox`. A `button` object is scripted to call a function named `isChecked()`. The argument `this.form` passes the form's name/value pairs to the function.

11. **Editor:** In the `isChecked()` function that has been started for you, locate the comment that reads as follows:

```
// Create booleanChecked variable
```

Create a variable name `booleanChecked`. Assign as its value the return value of the `checked` property for `myCheckBox`. An `if` statement has been created for you that reflects to the user the state of the check box.

12. **Editor:** Save **lab7-1.2.htm**.

13. **Browser:** Open **lab7-1.2.htm**. Your screen should resemble Figure 7-4.

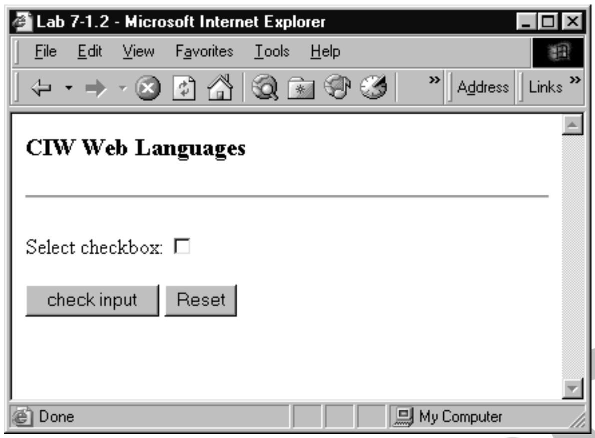


Figure 7-4: Lab7-1.2.htm

14. **Browser:** Test the page to ensure that the proper alert dialog box appears when the check box is selected. If the proper alert dialog box does not appear, verify that the source code you entered is correct.

This lab demonstrates some basic form field validation. You tested the value of a `text` object, and took appropriate action if the user did not enter any information. You saw that you were able to detect the state of a `checkbox` object in your script. You also used code that passed form data to a function using the `functionName(this.form)` syntax. In your script, you were then able to refer to a `text` element's value using the following syntax:

```
form.textElementName.value;
```

This syntax is a simpler alternative to the following:

```
document.forms[0].textElementName.value;
```

The *radio* Object

The `radio` object is used to select an option from among two or more mutually exclusive options. For example, if you ask users a yes-or-no question, you would want them to respond yes or no, but not both. Such a radio button pair could be written as follows:

INSTRUCTOR NOTE: The `length` property is the only `radio` object property that applies to all radio buttons in a group. Every other `radio` object property must be accessed with code directed to a specific radio button within the group.

```
<FORM NAME="radioForm">
Do you like apples?<BR>
<INPUT TYPE="radio" VALUE="yes" NAME="likeApples"> Yes
<INPUT TYPE="radio" VALUE="no"  NAME="likeApples"> No
</FORM>
```

Radio button groups are defined by the `NAME` property of each radio button. Radio buttons with the same name are in the same group, and only one button from each group can be selected at a time.

JavaScript accesses radio button values as if they were held in arrays. In JavaScript, you would refer to the value of the first button in the previous example as follows:

```
document.radioForm.likeApples[0].value;
```

You could display this value with the following statement:

```
alert(document.radioForm.likeApples[0].value);
```

The problem with the previous script is that it does not test whether the "yes" option (the first option) is checked. It returns the value of "yes" whether the option button was checked or not. The next lab demonstrates a function that determines which radio button was checked.

INSTRUCTOR NOTE: This section demonstrates how to detect if a `radio` object is selected. The students will apply this information in the next lab.

Note that the number of radio buttons to check can be determined using the `length` property of the `radio` object. A `for` loop is a convenient control structure for determining which `radio` object is selected in a group of `radio` objects. The following example uses the previously defined radio buttons:

```
var len = likeApples.length;

for (var i = 0; i < len; i++) {
  if (document.forms[0].likeApples[i].checked {
  // code to execute if radio object is checked
  }
}
```

Table 7-6 describes the commonly used properties, methods and event handlers related to the `radio` object.

Table 7-6: Features of `radio` object

Property	Method	Event Handler
`checked` Specifies or returns a Boolean value indicating `true` if the button is selected or `false` if the button is not selected	`click()` Simulates a user click on the radio button	`onClick` Specifies the JavaScript code to execute when the user clicks the radio button, or when the `click()` method is invoked
`defaultChecked` Returns a Boolean value indicating `true` if the `CHECKED` attribute was used in the `<INPUT>` tag for that `radio` object		
`form` Returns a reference to the `form` object containing the element		
`length` Returns an integer value indicating the number of `radio` objects in a group		
`name` Returns a string value of the name of this object as determined by the `NAME` attribute		
`value` Specifies or returns a string value determined by the `VALUE` attribute		

Lab 7-2: Using radio buttons

This lab will provide an opportunity to work with radio buttons in your JavaScript code.

1. **Editor:** Open the **lab7-2.htm** file from the Lesson_7 folder of the Student_Files directory.

2. **Editor:** This file contains an HTML form with a group of `radio` objects named `myRadio`. A `button` object is scripted to call a function named `isChecked()`. The argument `this.form` passes the form's name/value pairs to the function.

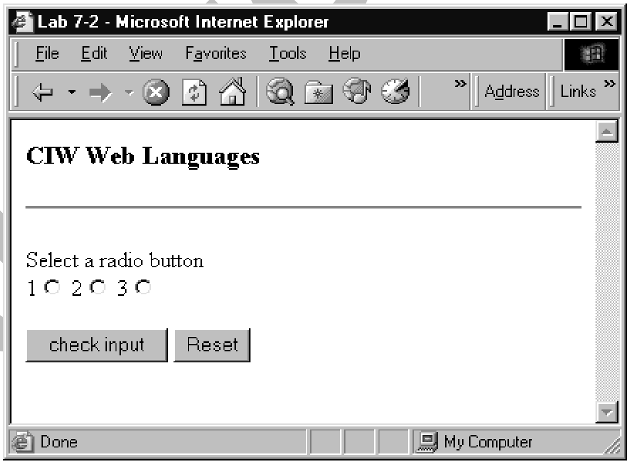
3. **Editor:** In the `isChecked()` function that has been started for you, a variable named `len` has been created. This variable is assigned the value returned by the `length` property for the `myRadio` group. Locate the comment that reads as follows:

   ```
   // Create for loop
   ```

   Create a `for` loop. Use a loop counter variable that starts at zero. Use the `len` variable in the loop's logical expression. Inside the loop, determine whether a particular radio object is selected. Create an alert dialog box that reflects to the user the value of any selected radio button.

4. **Editor:** Save **lab7-2.htm**.

5. **Browser:** Open **lab7-2.htm**. Your screen should resemble Figure 7-5.

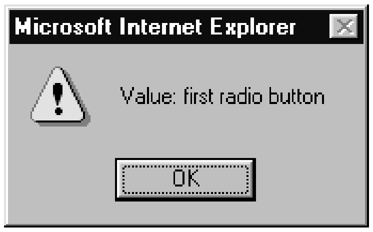

Lab 7-2 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
check input
Reset
Done
My Computer

CIW Web Languages

Select a radio button
1 2 3

Figure 7-5: Lab7-2.htm

6. **Browser:** Select a radio button, then click the **check input** button. If the first radio button is selected, you should see an alert similar to Figure 7-6. If you do not, verify that the source code you entered is correct.



Value: first radio button

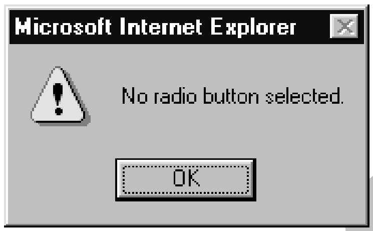
Figure 7-6: Alert dialog box

7. **Browser:** Ensure that the proper message is received for each radio button selection.

8. **Editor:** Save **lab7-2.htm** as **lab7-2.1.htm**. Determine a way to give the user an appropriate message if no radio button is selected when the check input button is clicked.

9. **Editor:** Save **lab7-2.1.htm**.

10. **Browser:** Open **lab7-2.1.htm**. Your screen should resemble Figure 7-5. Click the **check input** button without selecting a radio button. You should see an alert dialog box as shown in Figure 7-7. If you do not, check the logic of your source code.



No radio button selected.

Figure 7-7: Alert dialog box

This lab demonstrates that the state of `radio` objects can be captured in your scripts. You saw that the value of the `length` property of a group of `radio` objects can be used in a `for` loop to detect which radio button is selected. You also created code that executed if no radio buttons were selected.

The *select* Object

INSTRUCTOR NOTE: Remind students that unlike the **onChange** event handler for the **`text`** object, the **`select`** object's **onChange** event handler executes associated code immediately after the user makes a selection from a drop-down menu.

The `select` object is a drop-down list or a list box of items used in an HTML form. A selection list allows you to choose one item from a list. A list box allows you to choose one or more items from a list.

These objects are useful in forms, and may also be useful as stand-alone items that provide the user a choice of background color, pages to visit, or possible answers for a multiple-choice exam. Like any other form element, you should be able to capture and refer to the user's choices in the `select` object.

If you need to execute code as the result of the user choosing an option from a `select` object, the commonly used event handler is `onChange`. Using this event handler avoids launching script every time a user accesses the list simply to look at it. In this case, an event occurs only if a user changes from the default option to some new option in the list.

The syntax for creating a `select` object is as follows:

```
<SELECT NAME="mySelectBox">
  <OPTION>First Option
  <OPTION>Second Option
  <OPTION>Third Option
</SELECT>
```

In the preceding example, the text associated with each option is the only indication of the user's selection. If needed, an optional `VALUE` attribute can be added in the `<OPTION>` tag as shown in the following example:

```
<SELECT NAME="mySelectBox">
  <OPTION VALUE="1">First Option
  <OPTION VALUE="2">Second Option
  <OPTION VALUE="3">Third Option
</SELECT>
```

If this is a multiple-selection list, you can modify the preceding code as follows:

```
<SELECT NAME="mySelectBox" MULTIPLE SIZE="3">
  <OPTION VALUE="1">First Option
  <OPTION VALUE="2">Second Option
  <OPTION VALUE="3">Third Option
</SELECT>
```

Methods and properties of *select*

The `select` object has several methods, such as `blur()` and `focus()`, although they are not commonly used. The `select` object has three event handlers: `onBlur`, `onFocus` and `onChange`. These event handlers are described later in this lesson.

INSTRUCTOR NOTE: Refer to **Instructor Note 7-3** for important information concerning JavaScript's ability to dynamically modify the options in a `select` object.

Of particular interest to JavaScript programmers is the `options` property. This property provides script access to the various options defined with the HTML `<OPTION>` tag. The `options` property is also considered an object and contains its own properties.

In the following table, the string `...formRef` represents the actual `form` and `select` object names. For example, consider the following reference:

```
document.myForm.mySelectField.length
```

This reference would be shortened as follows:

```
...formRef.length
```

Table 7-7 lists the properties and subproperties of the `select` object.

Table 7-7: Properties of `select` object

Property	Subproperty	Description
`length`		`...formRef.length` Specifies or returns an integer value indicating the number of options in the `select` object *Note: This property is writable only in Netscape Navigator 3 (and later) and Microsoft Internet Explorer 4 (and later).*
`name`		`...formRef.name` Specifies or returns a string value of the name of the `select` object as determined by the `NAME` attribute in the `<SELECT>` tag
`options`		`...formRef.options` String value of the actual HTML code from the beginning `<SELECT>` tag through the ending `</SELECT>` tag *Note: Netscape Navigator 6 returns the object reference `[object NSHTMLOptionCollection]` and Microsoft Internet Explorer 5.5 returns the object reference `[object]` for this property.*

Table 7-7: Properties of `select` object (cont'd)

Property	Subproperty	Description
	`length`	`...formRef.options.length` Specifies or returns an integer value indicating the number of options in the `select` object *Note: This property is writable only in Netscape Navigator 3 (and later) and Microsoft Internet Explorer 4 (and later).*
	`selectedIndex`	`...formRef.options.selectedIndex` Specifies or returns an integer value of the array number of the selected option
`options[i]` *Where `i` is the index number of the currently selected option*		
	`defaultSelected`	`...formRef.options[i].defaultSelected` Returns a Boolean value determined by whether the option represented was selected by default in the original `<OPTION>` tag using the `SELECTED` attribute
	`selected`	`...formRef.options[i].selected` Specifies or returns a Boolean value determined by whether the option represented is selected.
	`text`	`...formRef.options[i].text` Specifies or returns a string value of the text following the `<OPTION>` tag in the HTML code for the option represented
	`value`	`...formRef.options[i].value` Specifies or returns a string value representing the assigned `VALUE` attribute specified within the `<OPTION>` tag

Just as with the `radio` object, a `for` loop provides a convenient mechanism for determining the user's choice from a `select` object. In the following example, the return value from a `select` object's `length` property is used to determine how many times the loop will execute. This example extracts the text property associated with the user's selection.

INSTRUCTOR NOTE: This section demonstrates to the students how to determine the choice made from a `select` object. The students will apply this information in the next lab.

```
var len = document.myForm.myselect.length;

for(var i = 0; i < len; i++) {
  if(document.myForm.mySelect.options[i].selected) {
    var myValue = document.myForm.mySelect.options[i].text;
  }
  // Use myValue inside the loop, if needed
}
// Use myValue outside the loop, if needed
```

This example uses an `if` statement to test the `selected` property of the `options` object. Note the use of the loop counter variable as the subscript for the `options` object. Next, a variable named `myValue` is assigned the return value of the `text` property of the `options` object. That variable can then be used as needed in the rest of the program.

INSTRUCTOR NOTE: Encourage students to create a variable that is assigned the value of a `select` object's `length` property for use in `for` loops. This technique is more efficient than continually evaluating the `length` property inside the `for` loop's definition.

In the next two labs, you will work with single-selection and multiple-selection boxes. You will create script that will evaluate the user's choice in the option box and then reflect the choice or choices to the user.

Lab 7-3: Using a *select* object

1. **Editor:** Open the **lab7-3.htm** file from Lesson_7 folder of the Student_Files directory.

2. **Editor:** This file contains an HTML form with a `select` object named `mySelect`. A `button` object is scripted to call a function named `isSelected()`. The argument `this.form` passes the form's name/value pairs to the function.

3. **Editor:** In the `isSelected()` function that has been started for you, a variable named `len` has been created. This variable is assigned the value returned by the `length` property for the `mySelect` object. Locate the comment that reads as follows:

```
// Create for loop
```

   Create a `for` loop. Use a loop counter variable that starts at zero. Use the `len` variable in the loop's logical expression. Inside the loop, determine whether a particular option is selected. Create an alert dialog box that reflects to the user the `value` property of any selected option.

4. **Editor:** Save **lab7-3.htm**.

5. **Browser:** Open **lab7-3.htm**. Your screen should resemble Figure 7-8.

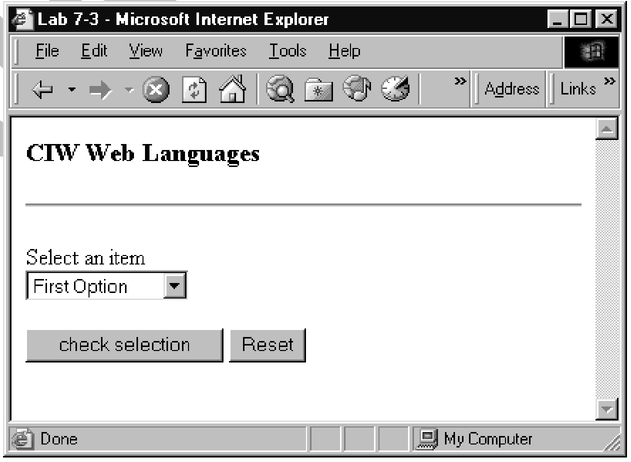


Figure 7-8: Lab7-3.htm

6. **Browser:** Make a selection from the drop-down menu and click the **check selection** button. If the second option is selected, you should see an alert dialog box that resembles Figure 7-9. If you do not, verify that the source code you entered is correct.

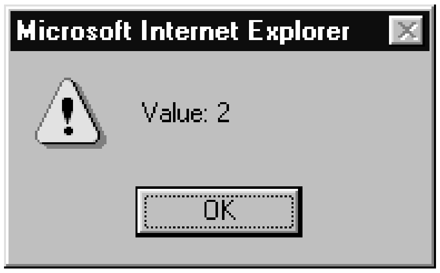


Figure 7-9: Alert dialog box

7. **Browser:** Test the page to ensure that the proper value is returned for each item in the drop-down menu.

8. **Editor:** Open **lab7-3.1.htm**. In the body of the document, locate the comment that reads as follows:

   ```
   <!-- Add onChange event handler -->
   ```

   This file is a modified version of lab7-3.htm. An extra `<OPTION>` tag is defined before the existing options. This option's text reads: *Select an item.* Also, no `button` object is scripted to invoke the function as in lab7-3.htm. In the HTML `<SELECT>` tag, add an `onChange` event handler that invokes the `isSelected()` function. Make sure to pass `this.form` as an argument in the method invocation statement.

9. **Editor:** After the `isSelected()` method invocation statement, add a semicolon. You will now add another statement that will also execute via the `onChange` event handler. Add this line of code:

   ```
   this.selectedIndex=0;
   ```

10. **Editor:** Save **lab7-3.1.htm**.

11. **Browser:** Open **lab7-3.1.htm**. Your screen should resemble Figure 7-10.

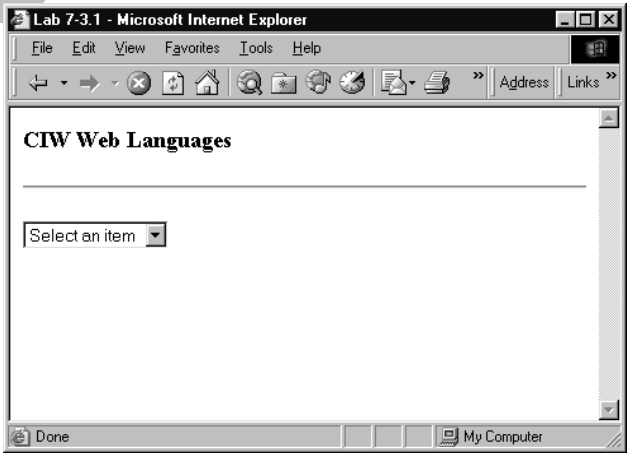

Lab 7-3.1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
CIW Web Languages
Select an item
Done
My Computer

Figure 7-10: Lab7-3.1.htm

12. Browser: Select an item from the drop-down menu. If the third item is selected, you should see an alert similar to Figure 7-11. If you do not, verify that the source code you entered is correct.

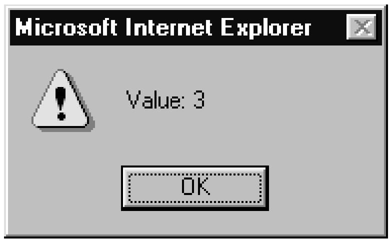


Figure 7-11: Alert dialog box

13. Browser: Note that the drop-down menu does not hold the user's selection after closing the alert dialog box. The drop-down menu should revert to its original state. If it does not, verify that the `this.selectedIndex=0` statement added in Step 9 is correct.

In this lab, the function `isSelected()` is scripted to receive the contents of the form on the page. Inside the function, you determined which option was selected from the drop-down menu and reflected the option's value to the user. As with all form elements, it is important to know how to access the value or text of an option chosen from a `select` object.

This lab also demonstrated the `onChange()` event handler used to invoke a function. The `onChange()` event handler was also used to invoke another line of JavaScript code. In this case, the `selectedIndex` property of the `select` object is used to reset the drop-down menu to its default state. Also, the programs were scripted to capture a single selection from a `select` object.

Multiple-Selection Lists

INSTRUCTOR NOTE: When the `MULTIPLE` attribute is present in a `select` object, the value of the `selectedIndex` property will reflect only the first, or topmost, option chosen from the list.

If the `MULTIPLE` attribute is used within the `<SELECT>` tag, the user can choose more than one option. Although various approaches can be taken to ascertain a user's multiple selections, a common solution is to test the `selected` property to determine the user's choices. Consider the following example:

```
<FORM NAME="myForm">
<SELECT NAME="mySelect" MULTIPLE SIZE="3">
    <OPTION>First Option
    <OPTION>Second Option
    <OPTION>Third Option
</SELECT>
</FORM>
```

INSTRUCTOR NOTE: This section demonstrates how to determine the choices made from a `select` object created with the `MULTIPLE` attribute. Students will apply this information in the next lab.

The following code captures the user's choices from the `mySelect` option list:

```
var len = document.myForm.mySelect.length;
var userChoices = "";

for (var i = 0; i < len; i++) {
  if (document.myForm.mySelect.options[i].selected) {
    //code to execute if option is selected
    userChoices += //either the option's text or value
  }
}
```

In this example, the `userChoices` variable is used to hold either the selection's text or values. Note the use of the += operator. This operator maintains any value assigned to the `userChoices` variable while adding any new data each time through the loop. Note that for the code in the previous example to function properly, code would have to be added that actually assigns values to the `userChoices` variable.

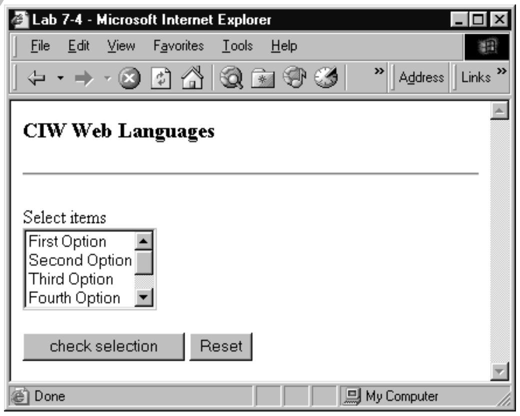
You will determine the user's choices from a multiple-selection list in the following lab.

Lab 7-4: Using a multiple-selection list

1. **Editor:** Open **lab7-4.htm** file from the Lesson_7 folder of the Student_Files directory.
2. **Editor:** This file contains an HTML form with a `select` object named `mySelect`. The select object is created as a multiple selection list. A `button` object is scripted to call a function named `isSelected()`. The argument `this.form` passes the form's name/value pairs to the function. In the `isSelected()` function, two variables are created for you. The `len` variable is used as in the previous lab. The variable `theSelections` will receive the text of the user's choices.
3. **Editor:** Locate the comment that reads as follows:

   ```
   // Create for loop
   ```

   Create a `for` loop. Use a loop counter variable that starts at zero. Use the len variable in the loop's logical expression. Inside the loop, determine whether a particular option is selected. If an option is selected, use the `theSelections` variable to build a string containing the text of the selected items. Add a newline character after the text for each selection.
4. **Editor:** After the `for` loop, create an alert dialog box that displays the `theSelections` variable.
5. **Editor:** Save **lab7-4.htm**.
6. **Browser:** Open **lab7-4.htm**. Your screen should resemble Figure 7-12.


Lab 7-4 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
Done
My Computer

CIW Web Languages

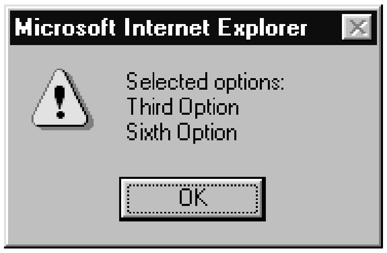
Select items

First Option
Second Option
Third Option
Fourth Option

check selection Reset

Figure 7-12: Lab7-4.htm

7. **Browser:** Select several items from the list, then click the **check selections** button. If the third and sixth items were selected, you should see an alert dialog box similar to Figure 7-13. If you do not, verify that the source code you entered is correct.



Selected options:
Third Option
Sixth Option

Figure 7-13: Alert dialog box

8. **Browser:** Experiment with the page to ensure that the alert dialog box displays the proper information in various situations.

The previous labs demonstrated methods that can be used to capture a user's choices from single-selection and multiple-selection lists. As previously mentioned, a critical aspect of using JavaScript with HTML forms is the ability to determine the user's input or selections as the user interacts with various form elements.

Form Validation

INSTRUCTOR NOTE: Refer to **Instructor Note 7-4** for additional information concerning JavaScript and form validation.

One of the most common and functional uses of client-side JavaScript is to validate form field submissions. As mentioned previously, it is not possible to completely validate all form submissions; however, some validation is better than none. Some benefits of client-side validation are:

- Increased validity of form submissions.
- Increased end-user satisfaction.
- Conservation of bandwidth.

Client-side form validation should be performed whenever possible. However, you must use caution. For example, end users may be annoyed by intervention from the program. Finding the right balance of program interaction takes thoughtful programming and thorough application testing.

An optional lab for this lesson demonstrates client-side form validation.

Lesson Summary

Application project

This project will challenge you to use some of what you have learned in this lesson.

This small application will present the user with an HTML page containing a form that simulates a credit card number submission form. This form has been created for you and is named **lesson7AppProj.htm**. This file is located in the Lesson_7_Application_Project folder in the Lesson_7 directory.

Your task is to add script to this file so that when a button is clicked, a script will perform validation tests. For instance, test with script to see whether the expiration date is valid. If it is not, prompt the user to update that field in the form.

Test the digits entered in the Card Number field to see whether they are valid for the type of credit card selected at the top of the form. For example, Visa card numbers always begin with the numeral 4 and use 16 digits. MasterCard numbers always begin with the numeral 5 and use 16 digits. American Express card numbers always begin with the numeral 3 and use 15 digits. If the numbers are incorrect based on the card type selected, ask the user to either select a different credit card or change the card number. If the number of digits corresponds correctly, acknowledge that.

Note that you are not expected to create an algorithm sophisticated enough to actually verify credit card numbers. However, creating the logic to perform the task previously described will strengthen your understanding of JavaScript when used to interact with user-submitted information.

Skills review

The JavaScript object model provides programmatic access to all elements that comprise HTML forms. The `form` object represents an HTML form in JavaScript. The `form` object is used to access all properties of a form and also provides methods to work with forms. The `button` object provides a scriptable, push-button type of user interface element on an HTML page. A `checkbox` object is an input object that provides properties, methods and event handlers to work with HTML check boxes. The `text` object displays a single line of text and provides programmatic access to user-entered values. The `radio` object is used to select an option from among mutually exclusive options. JavaScript can be used to detect and respond to the option that a user has selected. The `select` object is a drop-down list or a list box of items from which one item or multiple items can be selected. As with other form elements, your JavaScript programs can detect and use the selections that the user chooses with the `select` object. One of the most common and functional uses of client-side JavaScript is to validate form field submissions. This lesson ended with an example of form validation using JavaScript.

Now that you have completed this lesson, you should be able to:

- ✓ Identify and use form controls.
- ✓ Refer to `form` objects.
- ✓ Define the `form` object.
- ✓ Use the `button` object.
- ✓ Use the `checkbox` object.
- ✓ Evaluate text in the `text` and `textarea` objects.
- ✓ Process `radio` object options.
- ✓ Capture choices from a select list.
- ✓ Conduct form validation.

Lesson 7 Review

1. Why are forms important to Web developers?

   Forms are the primary user interface in Web pages because they allow a Webmaster to retrieve and verify data from users.

2. Name at least three form elements.

   `Button`, `checkbox`, `hidden`, `password`, `radio`, `reset`, `select`, `submit`, `text`, and `textarea`.

3. When you use form elements, what important requirement makes the form appear on the Web page in some browsers?

   To enclose all the form components within a pair of opening and closing `<FORM>` tags.

4. In what two ways can you refer to a `form` object's element?

   By its name, or by its index number in the `form` object's `elements` array.

5. What does the `form` object represent in JavaScript, and what is its purpose?

   The `form` object represents an HTML form in JavaScript. It is used to access all properties of a form, including the form's elements, and also provides methods to work with forms.

6. What is the main event handler of the `button` object? Which property specifies the text that will appear on the button?

   The main event handler of the `button` object is `onClick`. The `value` property specifies the text that will display on the button.

7. What is the main property associated with a `checkbox` object? What value does this property return?

   The main property of the `checkbox` object is `checked`. The `checked` property returns a Boolean value of `true` (if the box is submitted checked) or `false` (if the box is submitted unchecked).

8. How does the `text` object differ from the `textarea` object? How are they similar?

   The `text` object displays a single line of text, whereas a `textarea` object can display multiple scrolling lines of text. They use the same properties, methods and event handlers.

9. For what type of user input is the `radio` object best suited?

   The `radio` object is best suited for yes-or-no questions, or situations in which you want the user to choose only one response out of a group.

10. What event handler is commonly used to execute code as the result of the user choosing an option from a `select` object?

    The `onChange` event handler is most commonly used because it avoids launching script every time a user accesses the list simply to look at it.

11. What is one of the most common and functional uses of client-side JavaScript? What are some of the benefits of this use?

    To validate form field submissions. Some benefits of client-side validation include increased validity of form submissions, increased end-user satisfaction, and conservation of bandwidth.

Lesson 7 Instructor Section

This section is a supplement containing additional tasks for students to complete in conjunction with the lesson. It also contains additional instructor notes. The instructor may use all, some or none of these additional tools, as appropriate to the specific learning environment. These elements are:

- **Additional Instructor Notes**
  Detailed instructor notes that expand the tips and suggestions presented in the instructor margin notes throughout the lesson.
- **Optional Labs**
  Computer-based labs to be completed during class or as homework.
- **Lesson Quiz**
  Multiple-choice test to assess student knowledge of lesson material.

Additional Instructor Notes

The following section contains detailed instructor notes that expand the tips and suggestions presented in the instructor margin notes throughout the lesson.

Instructor Note 7-1

Location: The checkbox *Object*

Some students may ask how to programmatically select a `checkbox` object. In the following example, a form contains a check box named `myCheckbox`:

```
document.forms[0].myCheckbox.checked = true;
```

When executed, this line of code would select the check box. Note that the action caused by this line of code does not trigger a `click` event or invoke a `click()` method for `myCheckbox`.

Instructor Note 7-2

Location: Using a text box, a check box and a button

Students often ask whether the cursor can be placed at a specific location within a `text` object when using the `focus()` method. Normally, the cursor will appear at the beginning of the `text` object. However, in Microsoft Internet Explorer 4.0 and later, the `TextRange` object can be used to position the cursor at a specified location in a `text` object.

Also, in many applications, the `focus()` and `select()` methods are used in tandem to place the cursor in the `text` object while selecting any existing text that may be present. When using these two methods together, make sure to use the `focus()` method to first place focus on the `text` object, then use the `select()` method to select any text. Note that in newer browsers, using the `select()` method alone will place focus on the `text` object and select any text. However, this is not true of older browsers. It is always safest to use both methods when placing focus on a `text` object and selecting its text.

Instructor Note 7-3

Location: The select *Object*

Some students may be interested in the fact that JavaScript can dynamically add new contents to a `select` object. To add new options, the `new Option` constructor is used. Consider the following example:

```
<SCRIPT LANGUAGE="JavaScript">
<!-

  function addNewOptions(form) {
    var myText = prompt("Enter text for new option:","");
    var myValue = prompt("Enter value for new option:","");
    var len = form.mySelect.length;
    var myNewOption = new Option(myText, myValue);
    form.mySelect.options[len] = myNewOption;
  }

//-->
</SCRIPT>

</HEAD>
```

```
<BODY>
<H3>CIW Web Languages</H3>
<HR>

<FORM NAME="myForm">
<SELECT NAME="mySelect">
  <OPTION VALUE="1">Option 1
  <OPTION VALUE="2">Option 2
  <OPTION VALUE="3">Option 3
</SELECT>
<P>
<INPUT TYPE="button" VALUE="add new options"
onClick="addNewOptions(this.form)">
</FORM>
```

In this example, the user is prompted for the text and the values for new `select` object options. The `new Option` constructor receives these values, held in the `myText` and `myValue` variables, as its arguments. The `new Option` constructor can receive four arguments: the option's `text` property, the option's `value` property, a Boolean value that sets the option's `defaultSelected` property, and a Boolean value that sets the option's `selected` property. Following is an example supplying all four arguments to the `new Option` constructor.

```
var myNewOption = new Option(myText, myValue, false, true)
```

This example sets the `defaultSelected` property to false, and the `selected` property to true.

This line of code:

```
form.mySelect.options[len] = myNewOption;
```

adds the new option to the `select` object. Note the use of the `select` object's `length` property to determine where in the list the new option is placed. The original `select` object has three options, indexed `0` through `2`. The `length` property returns `3` in this situation. So as new options are added, the `len` variable always holds a new, unused index number for the new option.

Note that it is also possible to dynamically change existing options. To do so, the appropriate subscript is referenced when assigning the new option. Consider the following example.

```
form.mySelect.options[0] = myNewOption;
```

This line of code would dynamically change the existing text and value for the first `<OPTION>` tag defined within the select object.

Instructor Note 7-4

Location: Form Validation

Remind students that properly designed HTML forms can reduce the amount of work needed to validate form input. As a simple example, consider the following HTML form:

```
<FORM NAME="myForm" METHOD="post"  ACTION=http://ss1.prosofttraining.com/cgi-
bin/process.pl">

Name: <INPUT TYPE="text" NAME="name"><BR>
Date: <INPUT TYPE="text" NAME="date"><BR>

<INPUT TYPE="button" VALUE="Submit"
onClick="validateForm(this.form);">

</FORM>
```

This form asks the user for a name and a date. Validating the `name` element input would typically involve ensuring that at least some input existed. However, suppose the developer needed to ensure that the `date` element input was in a particular format. As an example, suppose the `date` element input needed to be in the format mm/dd/yyyy. In other words, the month must be entered first in a two-digit format. Then the day of the month must be entered in a two-digit format. Finally the year must be entered in a four-digit format. The JavaScript code needed to validate the `date` element input would be somewhat complex and would require significant development time. However, if the form is designed differently, the task becomes considerably easier. Examine the following HTML form:

```
<FORM NAME="myForm" METHOD="post"  ACTION=http://ss1.prosofttraining.com/cgi-
bin/process.pl">

Name: <INPUT TYPE="text" NAME="name"><BR>
Select Date: <BR>

Month:
<SELECT NAME="month">
<SCRIPT LANGUAGE="JavaScript">
<!–

  for(var i = 1; i <= 12; i++) {
    if(i <= 9) document.write("<OPTION>0" + i);
    else document.write("<OPTION>" + i);
  }

//-->
</SCRIPT>
</SELECT>

Day:
<SELECT NAME="day">

<SCRIPT LANGUAGE="JavaScript">
<!–

  for(var i = 1; i <= 31; i++) {
    if(i <= 9) document.write("<OPTION>0" + i);
    else document.write("<OPTION>" + i);
  }

//-->
</SCRIPT>
</SELECT>

Year:
<SELECT NAME="year">
<SCRIPT LANGUAGE="JavaScript">
<!–

  for(var i = 2001; i <= 2020; i++) {
    document.write("<OPTION>" + i);
  }
```

```
//-->
</SCRIPT>
</SELECT>
<P>
<INPUT TYPE="button" VALUE="Submit"
onClick="validateForm(this.form);">

</FORM>
```

The design of this form allows the user to select date information from drop-down menus. The user has no chance to enter the date information in any format other than that offered by the form. Note that the user can still submit errorneous information, and the format of the form submission may require some handling on the server end. However, the main point is to carefully design HTML forms to reduce the chance of user error and to facilitate form validation.

Optional Lab 7-1: Conducting form validation

In this lab, you will use JavaScript to validate user input to an HTML form.

1. **Editor:** Open the **optionalLab7-1.htm** file from the Lesson_7 folder of the Student_Files directory. The HTML code is already included to create a form that resembles Figure OL7-1.

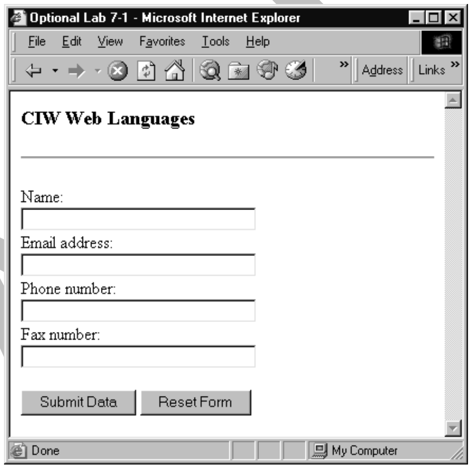


Figure OL7-1: Interactive form

2. **Editor:** A `sumbit` object is scripted to call a function named `checkForm()`. Examine the function invocation statement:

```
<INPUT TYPE="submit" VALUE="Submit Data"
onClick="return checkForm(this.form);">
```

3. **Editor:** Note the use of the `return` keyword. The `checkForm()` function will be written to return either a true or a false value. If a `true` value is returned, the form will be submitted. A `false` return value will cancel the form submission. The argument `this.form` passes the form's name/value pairs to the function

4. **Editor:** In the `checkForm()` function, a variable is created for you. The `len` variable is assigned the return value of the form object's `length` property. This variable will be used in a `for` loop to determine how many times the loop will execute.

5. **Editor:** Locate the comment that reads as follows: `// Create for loop`. Create a `for` loop. Use a loop counter variable that starts at zero. Use the `len` variable in the loop's logical expression. Inside the `for` loop, create an `if` statement. As the condition for the `if` statement, use the `elements` property of the `form` object to determine if each form element is either `null` or empty. Use the logical OR operator (||) for this task.

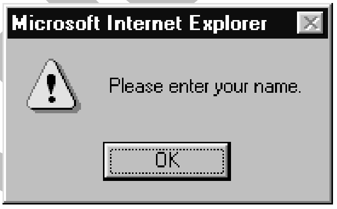
6. **Editor:** If the form element is either null or empty, create an alert dialog box that asks the user to enter information in the appropriate form element. *(Hint: Use the `name` property of each element in the argument for the `alert()` method.)* Then use the `focus()` method to place the user's cursor in the appropriate form element. Finally, create a `return false` statement. This statement will be returned to the function invocation statement and will cancel submission of the form. Close the `if` statement. Close the `for` loop.

7. **Editor:** After the `for` loop, create a `return true` statement. This statement will execute only if all form elements pass the validation test.

8. **Editor:** Another function named `emailTest()` has been created for you. This function will be discussed following this lab.

9. **Editor:** Save **optionalLab7-1.htm**.

10. **Browser:** Open **optionalLab7-1.htm**. Your screen should resemble Figure OL7-1. Click the **Submit Data** button without entering any data in the form. You should see an alert dialog box similar to Figure OL7-2. If you do not, verify that the source code you entered is correct.


Microsoft Internet Explorer

Please enter your name.

Figure OL7-2: Alert dialog box

11. **Browser:** After closing the alert dialog box, your cursor should be in the Name text box. If it is not, verify that the source code you entered is correct.

12. **Browser:** Experiment with the form to ensure that the proper alert dialog boxes appear for each situation. Ensure that the user's cursor is in the proper text box if the text box is left empty.

The preceding lab demonstrated client-side form field validation. You learned to test all the elements of a form and detect whether they held no value (or a `null` value).

The `checkForm()` function is scripted to test the values of the elements within the form. If any of the form elements do not pass the validation test, a `false` value is returned to the calling statement.

The `checkForm()` function receives the contents of the form into the `form` parameter. The variable `len` is declared and instantiated with the return value of the `form.elements.length` property. A `for` loop is created using the variable `len` to determine the number of loops. Inside the `for` loop, an `if` statement tests each form element for an empty or `null` value. An alert dialog box notifies the user if a field does not pass the test. The empty field then receives focus. The `return false` statement is then returned to the calling statement, thus negating the form submission. If all form elements pass the test, the `return true` statement returns a `true` value to the calling statement, and the form is submitted.

This lab also demonstrates code that tests for proper e-mail address format. In the `emailTest()` function, a pattern is created to test the submitted string against the normal e-mail address format of `userName@host.domain`. Note the way in which the `emailTest()` function is invoked:

```
<INPUT TYPE="text" SIZE="30" NAME="email address" onBlur="emailTest(this);">
```

The `onBlur` event handler is used to call the `emailTest()` function. When the user tabs away from or otherwise removes focus from the `email address` element, the function is called. Note that the keyword `this` is passed as an argument to the function. This action has the effect of passing data for the `email address` element to the function. Examine the `emailTest()` function:

```
function emailTest(emailText) {
  var email = emailText.value;
  var emailPattern = /^.+@.+\..{2,}$/;
    if (!(emailPattern.test(email))) {
      alert("Please enter a valid email address.");
      form.elements[1].focus();
     }
}
```

The `emailText` parameter receives data from the `email address` element. The variable `email` is declared and instantiated with the value of the `email address` element. Next, a pattern is created. Examine this pattern carefully:

```
/.+@.+\..{2,}/;
```

The period and the plus sign (`.+`) indicate that any character except a new line must appear one or more times. The literal text @ specifies that the @ symbol must appear next. This text is followed by another `.+` indicating that any character must appear one or more times. This syntax is followed by `\.` specifying that the pattern must continue with a literal period. Note that the period is a regular expression special character and must be escaped when matching a literal period in a pattern. Next is `.{2,}` specifying that the pattern must then contain any character at least two or more times. The comma after the `2` in `{2,}` indicates two or more characters instead of exactly two. Now consider the following e-mail address:

```
jdoe@company.com
```

An e-mail address such as this would match the pattern. The following e-mail address would also match the pattern:

```
jdoe@company.fr
```

Finally, an `if` statement is created. The `test()` method of the `RegExp` object tests the submitted string against the pattern held in `emailPattern`. Note the use of the logical NOT operator (`!`). In pseudo-code, this section of the function would read as follows: "If the e-mail address does not match the pattern, execute the alert dialog box, then place focus back in the e-mail address text box."

Note that the regular expression described in the preceding example will not verify all possible e-mail addresses. In fact, the pattern is very permissive because it allows characters that would not be allowed in an e-mail address. The regular expression needed to test all possible e-mail addresses are quite complex and are beyond the scope of this course.

Lesson 7 Quiz

1. Which example shows proper JavaScript syntax for referring to a `form` object?

   a. *`document.newForm.fieldName.value;`*
   b. `window.newForm.fieldName.value;`
   c. `form.newForm.fieldName."value"`
   d. `document.newForm[fieldName].value;`

2. In JavaScript, what is the purpose of the `form` object?

   a. To enclose the HTML `<FORM>` tags
   b. To represent all the forms in a single Web site
   c. To access all properties of a form
   d. To replace the HTML `<FORM>` tags

3. Which JavaScript object displays a single line of alphanumeric values and provides programmatic access to these user-entered values?

   a. The `textarea` object
   b. The `submit` object
   c. The `select` object
   d. The *`text`* object

4. Which JavaScript object allows you to choose one or more items from a list?

   a. The *`select`* object
   b. The `checkbox` object
   c. The `radio` object
   d. The `textarea` object

5. Which JavaScript input object can a user click to return a Boolean value of true or false?

   a. The `reset` object
   b. The *`checkbox`* object
   c. The `button` object
   d. The `select` object

6. Which JavaScript object provides a basic push-button interface and relies on scripting for its functionality?

   a. The `form` object
   b. The `submit` object
   c. The *`button`* object
   d. The `select` object

7. Which JavaScript object is used to select an option from among mutually exclusive options?

   a. The `select` object
   b. The `button` object
   c. The `checkbox` object
   d. <u>The *`radio`* object</u>

8. Which of the following is an event handler used by the `form` object?

   a. <u>*`onReset`*</u>
   b. `submit()`
   c. `onClick`
   d. `select()`

9. Which attribute defines a group of radio buttons for the `radio` object?

   a. `checked`
   b. <u>*`NAME`*</u>
   c. `TYPE`
   d. `value`

10. Which attribute is used to allow the user to choose more than one choice from a `select` object?

    a. `options`
    b. `SELECT`
    c. `length`
    d. <u>*`MULTIPLE`*</u>

11. Which of the following best describes the main purpose of client-side form validation with JavaScript?

    a. To verify that all user-entered data is true and correct
    b. To correct typing errors in user-entered data before processing
    c. <u>*To detect whether form elements hold no or a null value or properly formatted values*</u>
    d. To impress users with intervention from the program

12. Write a `<FORM>` tag that uses inline scripting to call a function named `clear()` when the form is reset.

    <u>*`<FORM METHOD=method ACTION=action onReset="clear();">`*</u>

13. Write a JavaScript function that writes true or false on a `button` object (named `myButton` in a form named `myForm`) depending on the result of a confirm dialog box.

    <u>*One way to write this code is as follows:*</u>

```
function changeButtonValue() {
  document.myForm.myButton.value = confirm("True or False  on the ➘
button?");
}
```

14. Write JavaScript code to create a function that checks a `text` object (named `myText` in a form named `myForm`) for an empty value. If there is no value, ask the user to enter a value and place the cursor back in the `text` object.

    One way to write this code is as follows:

```
function checkText() {
if(document.myForm.mytext.value = "") {
  alert("Please enter a value.");
  document.myForm.myText.focus();
  }
}
```

    Another way to write this code is as follows (the form element's values are passed to the function when the function is called):

```
function checkText(form) {
if(form.mytext.value = "") {
  alert("Please enter a value.");
  form.myText.focus();
  }
}
```

15. Write JavaScript and HTML code to create a `text` object (named `myText` in a form named `myForm`) that contains the read-only string *CIW Certified*.

```
<INPUT TYPE="text" NAME="myText" VALUE="CIW Certified"
onFocus="this.blur();">
```

16. Write JavaScript and HTML code to create a `checkbox` object that, when clicked, displays its state (checked or not checked) and value in an alert dialog box.

```
<INPUT TYPE="checkbox" VALUE="myValue" NAME="myCheckbox"
onClick="alert('The check box\'s state is ' + this.checked + 'with
a value of ' + this.value + '.');">
```

17. Consider the following code:

```
<FORM NAME="myForm">
value1<INPUT TYPE="radio" NAME="myRadio" VALUE="test1">
value2<INPUT TYPE="radio" NAME="myRadio" VALUE="test2">
value3<INPUT TYPE="radio" NAME="myRadio" VALUE="test3">
</FORM>
```

    Write a function that determines which `radio` object was selected, with the selected `radio` object's value reflected back to the user in an alert dialog box.

    One way to write this code is as follows:

```
function checkRadio () {
  if(document.myForm.myRadio[0].checked) {
    alert(document.myForm.myRadio[0].value);
  }
  if(document.myForm.myRadio[1].checked) {
    alert(document.myForm.myRadio[1].value);
  }
  if(document.myForm.myRadio[2].checked) {
    alert(document.myForm.myRadio[2].value);
  }
}
```

    A better way to write this code is as follows (the form element's values are passed to the function when the function is called):

```
function checkRadio(form) {
```

```
var num = form.myRadio.length;
  for(i = 0; i < num; i++) {
    if(form.myRadio[i].checked) {
      alert(form.myRadio[i].value);
      break;
    }
  }
}
```

18. Consider the following code:

```
<FORM NAME="myForm">
<SELECT NAME="mySelect">
<OPTION VALUE="value1">value1
<OPTION VALUE="value2">value2
<OPTION VALUE="value3">value3
</SELECT>
</FORM>
```

Write a JavaScript function that determines which `select` object's option was chosen, with the chosen `select` option's value reflected back to the user in an alert dialog box.

One way to write this code is as follows:

```
function checkSelect() {
  if(document.myForm.mySelect.selectedIndex == 0) {
    alert(document.myForm.mySelect.options[0].value);
  }
  else if (document.myForm.mySelect.selectedIndex == 1) {
    alert(document.myForm.mySelect.options[1].value);
  }
  else {
  alert(document.myForm.mySelect.options[2].value);
  }
}
```

A better way to write this code is as follows (the form element's values are passed to the function when the function is called):

```
function checkSelect(form) {
  var num = form.mySelect.length;
  for(i = 0; i < num; i++) {
    if(form.mySelect.options[i].selected) {
      alert(form.mySelect.options[i].value);
      break;
    }
  }
}
```

Lesson 8: Cookies and JavaScript Security

Objectives

By the end of this lesson, you will be able to:

- Explain cookies.
- Delete cookies from your disk.
- Assign a cookie.
- Test for the presence of a cookie.
- Clear a cookie.
- Enable and disable cookies in the browser.
- Use cookies and passwords to restrict entry to a page.
- Discuss security issues relevant to JavaScript.
- Define signed scripts.

Pre-Assessment Questions

1. Which of the following JavaScript cookies properly scripts a URL for cookie access?

   a. `document.cookie = "myCookie = myValue; url = /";`
   b. `document.cookie = "myCookie = myValue; pathName = /";`
   c. *<u>`document.cookie = "myCookie = myValue; path = /";`</u>*
   d. `document.cookie = "myCookie = myValue; access = /";`

2. Which of the following JavaScript cookies properly scripts an expiration date?

   a. *<u>`document.cookie = "myCookie = myValue; expires = Friday, 26-Nov-2010 06:00:00 GMT";`</u>*
   b. `document.cookie = "myCookie = myValue; expiration = Friday, 26-Nov-2010 06:00:00 GMT";`
   c. `document.cookie = "myCookie = myValue; expirationDate = Friday, 26-Nov-2010 06:00:00 GMT";`
   d. `document.cookie = "myCookie = myValue; terminates = Friday, 26-Nov-2010 06:00:00 GMT";`

3. Describe the best mechanism for deleting a JavaScript cookie.

   <u>The best way to delete a JavaScript cookie is to assign the cookie an expiration date that is in the past. This causes the cookie to expire and to be deleted from the user's computer.</u>

Security and Cookies in JavaScript

One of the reasons JavaScript is so popular is its ability to assign cookies. JavaScript is also a fairly secure language, although you need to consider the security problems it may cause. In this lesson, you will learn about cookies, how to control their use on your computer, and how to assign them. More importantly, you will learn the reasons that cookies are used so often, and you will consider some of the security issues involved with cookies. You will also learn about "helper" applications launched by JavaScript, and how to avoid coding infinite loops.

What Are Cookies?

cookie
Information sent between a server and a client to help maintain state and track user activities. Cookies can reside in memory or be placed on a hard drive in the form of a text file.

INSTRUCTOR NOTE: The Netscape specification for cookies is located at *home.netscape.com/newsref/std/cookie_spec.html*. Additional information is available at *portal.research.bell-labs.com/~dmk/cookie.html*.

Cookies are small pieces of information sent from a server to a client's computer and stored in memory. A cookie is originally stored in the memory of the client's computer, but the cookie can be scripted to store its information permanently on the hard drive, which has become prevalent. Almost every commercial Web site uses cookies; one site can deposit several of these files on your hard drive upon just one visit. Cookies are often referred to as "persistent cookies" and even "persistent HTML." In fact, Netscape created the cookie specification, which refers to cookies as "persistent client-state HTTP cookies."

The term "cookie" was chosen at random and has no special meaning. One possible explanation is that it is a slang term for a small token used to denote a transaction. For example, when you go to a movie and pay for admission, the ticket you receive is a type of "cookie" that you can use to prove you have permission to enter. Another explanation could be that UNIX programs contain operators, or keywords, called "magic cookies." These cookies help to track a user's operation of a specific program (such as storing and retrieving the data a user generated with the program, and so forth).

Whether stored in memory or in small text files on your hard drive, cookies have a variety of uses. They can help maintain state, aid authentication, remove redundant steps, and track user behavior. You will learn more about each of these uses in this lesson.

How Are Cookies Sent?

Cookies are sent from a server to the client's browser in the HTTP response header. The browser accepts the response from the page assigning the cookie, and then accepts (or gives the user the chance to accept or reject) the cookie. After the cookie has been accepted, the browser program stores the cookie information in system memory or, if the cookie is scripted to do so, in files.

Tech Note

You cannot assign cookies with Internet Explorer 3.0 unless you are running "live" on a Web server. You can, however, assign cookies with Navigator and Internet Explorer 4.0 with or without a server present. The labs in this lesson will not work on Internet Explorer 3.0 unless you are either running on a server or using a server product (such as the Personal Web Server) and accessing the page via the HTTP addressing protocol.

When a user generates an HTTP request, two actions can occur: Any cookies already on the client's system that match the server's domain can be passed along in the request header, or a server can send a cookie back in the HTTP response header. In subsequent exchanges between the client and the server, the server can then test and evaluate the contents of any cookies that may be present.

A cookie header appears to your browser as follows:

```
Set-Cookie: name=value; expires=date; path=path; domain=domain; secure
```

Each cookie header contains a set of parameters. Not all of these parameters must be assigned to use cookies. The cookie header parameters include the following:

- ***name=value***
  The name of the cookie and its value are set by this expression. The *name=value* pair is the only information required to generate a cookie. All other parameters are optional.
- **expires=*date***
  The expires=*date* pair determines when the cookie will expire. The date is formatted as follows:

  ```
  weekday, DD-Mon-YY HH:MM:SS GMT
  ```

  GMT stands for Greenwich Mean Time. The time expiration scheme is specified in RFCs 822, 850, 1036 and 1123.

  If the expires attribute is not used, the cookie will expire at the end of the user's session with the Web application.
- **path=*path***
  The path=*path* pair contains path information for the URL that issued the cookie. This attribute defines subsets of the URL for which the cookie is valid. The most commonly used path is / (forward slash), used to denote the root directory of the server.

  If the path attribute is not used, the URL path of the document setting the cookie is used.

INSTRUCTOR NOTE: The domain attribute can safely be omitted unless the HTML document containing the cookie will be duplicated on another server within a particular domain.

- **domain=*domain***
  The domain=*domain* pair contains the domain name of the URL that issued the cookie. When an HTTP request header is formed, the path and domain name pairs are checked; if they match the page being requested, any cookies pertaining to that domain are passed back to the server for evaluation. A cookie in a client's request header appears to the server as follows:

  ```
  Cookie: name=value; name2=value2; etc.
  ```

  The server does not see any information other than the name=*value* pairs. Only URLs matching the path and domain values in the cookie file can be read again and evaluated by the server.

  If the domain attribute is used, the server setting the cookie must be part of the domain being set in the cookie. Thus, a server named *www.ThisServer.com* cannot set a cookie for the domain *www.ThatServer.com*.

  If the domain attribute is not used, the domain of the document setting the cookie is used.
- **secure**
  If the secure parameter is present, the cookie will only be sent using a secure protocol, assuming one is available. If the secure attribute is not used, the cookie is deemed safe to be sent over unprotected channels.

Who Can Send Cookies?

The general guideline for cookies is that a server can set, or deposit, a cookie only if a user visits that particular site. In other words, one domain cannot deposit a cookie for another, and (in theory) cross-domain posting is not possible. Additionally, a server can retrieve only those cookies it has deposited; one server cannot retrieve a cookie set by another.

Shared cookies

A more recent technique for using cookies includes several domains uniting and creating a large network, thereby setting cookies on the hard drives of users who have never visited a particular site. By simply visiting one site, a user can become enrolled in a larger network through the setting of one cookie. This practice is called "sharing cookies."

For example, Yahoo! can join with another larger network such as DoubleClick.net, which allows fellow network members such as Amazon.com to load content onto its main search page. This content is often an image-based advertisement loaded with a Java applet. Because all of these sites essentially belong to the same network as subdomains, whenever a user accesses the Yahoo! main page, he or she is also accessing the Amazon.com site.

In addition to the image, however, the advertisement applet could also deposit a cookie from Amazon.com when the Yahoo! site loads into the browser. Therefore, even though a user has visited only Yahoo!, the cookie can be set from another site (Amazon.com). This practice is increasingly common, and it seemingly bypasses the rule concerning cross-domain posting.

Storing Cookies

Currently, a domain can store no more than 20 cookies on a user's computer, and a user can store no more than 300 cookies total on his or her hard drive. If this limit is reached, the browser will delete the least-used cookie. The HTTP header for a cookie can be no larger than 4 kilobytes.

Cookies store `name=`*`value`* pairs as text strings. Cookies can expire as soon as the user exits the site that issued the cookie, or they can be set to expire on some future date. Thus, cookies can persist until they expire or are deleted by the user.

Netscape Navigator and Microsoft Internet Explorer

Many different browsers exist, and each stores cookies differently. Additionally, different versions of the same program can store cookies in different directories. For the purposes of clarity, this lesson will discuss the two major browsers: Netscape Navigator and Microsoft Internet Explorer.

Netscape Navigator

Netscape Navigator always store cookies in a single text file named *cookies.txt*. Navigator Version 3.0 stores the file in the *systemroot*\Program Files\Netscape\Navigator\Program folder. Navigator 4.0 places its cookie file in the *systemroot*\Program Files\Netscape\Users\Username directory. Navigator 6.0 places its cookie file in the *systemroot*\Program Files\Netscape\Users50\Username directory.

Microsoft Internet Explorer

Regardless of version, Microsoft Internet Explorer stores cookies in individual text files. Internet Explorer 3.0 stores its files in the *systemroot*\Windows\Cookies directory. Version 4.0 stores cookies in the *systemroot*\Windows\Temporary Internet Files* directory. Versions 5.x and 6.x store cookie files in the *systemroot*\Windows\Cookies directory.

INSTRUCTOR NOTE: The DAT files used by Microsoft Internet Explorer are cache files used to store URL information for Web sites visited with the IE browser. Copies of the files may also be stored in c:\Windows\History, c:\Windows\Temp, or c:\Windows\ Temporary Internet Files. These files will grow in size over time. If the DAT files are deleted, Windows will recreate smaller versions of them. Also, some versions of IE create a file named index.dat instead of mm2048.dat or mm256.dat.

Regardless of version, Internet Explorer uses two types of files when setting cookies. The first is the cookie text file. The name of each file is similar to an e-mail address in that it lists the user name of the browser, then the name of the domain that set the cookie (for example, *jdoe@ciwcertified.txt*). Again, the first part of the name is taken from the user name registered by the Microsoft Internet Explorer program. The second part of the name (after the @ sign) denotes the origin of the cookie file.

The second type of file is a DAT file (for example, mm2048.dat or mm256.dat). The DAT file is technically not a cookie; it is used by the browser to enable caching. However, Internet Explorer uses both the text and DAT files in tandem when working with cookies.

A user can delete or disable cookie files

Any knowledgeable user can delete or disable cookie files. A user can also write-protect cookie files and folders, as well as use a statement in the computer's autoexec.bat file that automatically deletes cookie files every time the machine boots. For this reason, you should thoroughly test any application that depends upon cookies for its functionality. You would need to include code that tests for the presence of cookies, and possibly provide alternative code for situations in which cookies cannot be used.

Why Use Cookies?

Cookies can be used for many reasons. You can use cookies on your Web site to provide authentication, store user information, or maintain user state.

Authentication

You can deploy cookies to aid user authentication. For example, if a user visits your Web site and properly authenticates, you can deposit a cookie on that user's hard drive. Then, when the user returns, he or she will no longer have to enter a password because that information is stored in a cookie on his or her hard drive.

Although this practice can provide a level of convenience, it can also present a hazard. Because the cookie is stored on that user's hard drive, any other user could sit down at that computer and instantly log on to the password-protected site and impersonate the authenticated user.

Storing user information

Using cookies, a Web site can gain the following information about a client:

- Operating system and browser type
- Service provider
- IP address
- A history of sites visited

State maintenance with cookies

INSTRUCTOR NOTE: Most applications that utilize cookies for state maintenance do so with cookies generated from a server-side script.

One of the most common uses of cookies is to store information about a user to maintain state. HTTP primarily uses TCP, which is a stateful (i.e., connection-oriented) protocol. However, most Web servers terminate these connections in a short time. It is also natural for a user to end a connection quickly.

To compromise for brief connections on either side of the Web browser, cookies allow you to remember connections over time. For example, suppose a user is playing an adventure game and decides to leave it. You can create a cookie that tells your site where that user left the game. When the user returns, he or she can resume at that point. Alternatively, a user can choose a color scheme for touring your site. You can also script pages to respond to cookie information so the user's color preferences follow his or her path.

In general, cookies have no right or wrong uses. You can use them for any purpose you want, but be prepared for the possibility that the user might reject or delete the cookie.

Saving time for return visitors

As a part of state maintenance, you can create cookies that eliminate certain steps. For example, during the registration process, Amazon.com deposits a cookie that allows a user to purchase a book by clicking only once. The user does not have to re-enter credit card or any other information.

Assigning a Cookie

The `document` object contains the `cookie` property to work with cookies in JavaScript. Assigning a cookie is quite simple. The syntax is as follows:

```
document.cookie = "name = value";
```

Alternatively, you can use the following syntax:

```
document.cookie = "name = value;expires = date";
```

INSTRUCTOR NOTE: This section demonstrates how to create a JavaScript cookie. Students will apply this information in the next lab.

For example, to create a cookie named `userName`, the following code could be used:

```
document.cookie = "userName = Raul";
```

The only parameter pair required is the *`name=value`* pair. If no expiration date is assigned, the cookie will be deleted when the browser is closed. If an expiration date is present, the cookie will expire on that date.

If you want to ensure that your cookie is sent over a secure connection (assuming one is available), you can add the `secure` parameter at the end, after another semicolon. For example:

```
document.cookie="name=value;expires=date;secure";
```

In addition, if you want to specify a particular domain and path name, you can add these parameters as well. However, be aware that when you want to evaluate the contents of a cookie, you will only be able to access cookies with a domain and path name that matches that of the server's response header.

Testing for Cookie Presence

INSTRUCTOR NOTE: When using Netscape Navigator or Microsoft Internet Explorer 4.0 and later, typing `javascript:alert(document.cookie);` into the address bar and pressing ENTER will display an alert dialog box showing all cookies associated with that HTML page.

You can easily test for the presence of any cookie by using the `document.cookie` statement in your script. For example:

```
<SCRIPT LANGUAGE="JavaScript">
<!--

alert(document.cookie);

//-->
</SCRIPT>
```

Each time a user accesses this page, he or she will see a quick listing of all cookies associated with that particular browser session in the following format:

`name1=value1,name2=value2,` etc.

If no cookies are present, the user will see an empty alert.

Generally, you should test for a specific cookie by name or by value to see whether it exists. Then you can extract the relevant information from the cookie.

Clearing a Cookie

To clear a cookie, simply reassign it, adding an expiration date that has already passed. If a cookie is assigned to an existing cookie name, the new cookie will replace the old one. By setting the date in the past, the cookie will expire and be deleted from the user's computer.

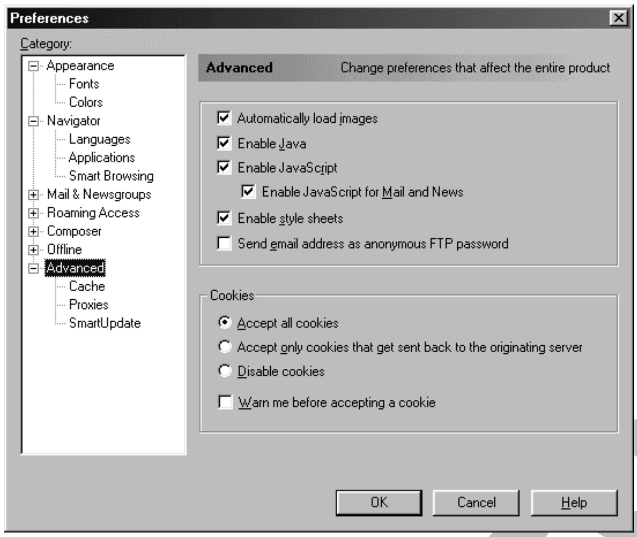
Controlling Cookies in the Browser

Any user can choose which cookies to accept by adjusting the settings in the browser. Following are two procedures: The first controls cookies in Netscape Navigator, and the second controls cookies in Microsoft Internet Explorer.

Netscape Navigator

Follow these steps to enable or disable cookies in Netscape Navigator 4.x. You can also customize your notification about incoming cookies.

1. From the main menu, choose Edit | Preferences.
2. In the Category window (to the left of the Preferences dialog box), select Advanced.
3. You should see the dialog box shown in Figure 8-1.



Advanced — Change preferences that affect the entire product

- [✓] Automatically load images
- [✓] Enable Java
- [✓] Enable JavaScript
  - [✓] Enable JavaScript for Mail and News
- [✓] Enable style sheets
- [ ] Send email address as anonymous FTP password

Cookies

- [● Accept all cookies]
- ○ Accept only cookies that get sent back to the originating server
- ○ Disable cookies
- [ ] Warn me before accepting a cookie

Figure 8-1: Preferences in Netscape Navigator 4.x

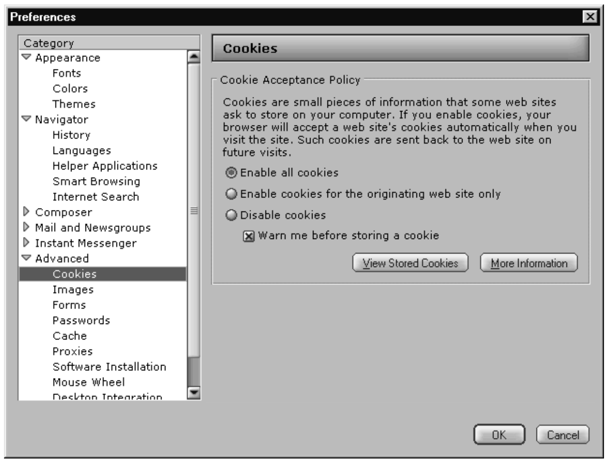
4. Select any one of the following options.

   - **Accept all cookies:** This option imposes no restrictions.
   - **Accept only cookies that get sent back to the originating server:** This option will disable shared cookies.
   - **Disable cookies:** Your browser will no longer accept any cookies, nor will it warn you of any.
   - **Warn me before accepting a cookie:** If you select this check box, you will be warned each time a server attempts to set a cookie. This option will warn you quite often.

For Netscape Navigator 6.x, use the following steps:

1. From the main menu, choose Edit | Preferences.
2. In the Category window (to the left of the Preferences dialog box), select Advanced | Cookies.
3. You should see the dialog box shown in Figure 8-2.

INSTRUCTOR NOTE: The information in this section is valuable to all end users. However, knowing how to control cookies in the browser is also important for JavaScript developers. Changing the browser's cookie settings allows the developer to properly test applications that use JavaScript cookies.



Cookies

Cookie Acceptance Policy

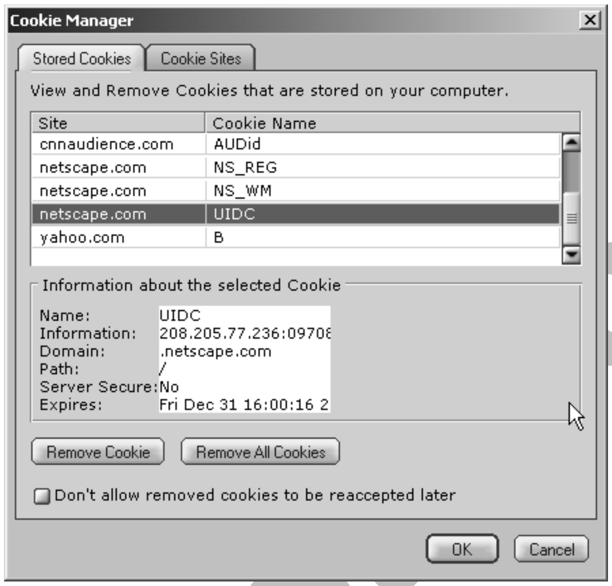
Cookies are small pieces of information that some web sites ask to store on your computer. If you enable cookies, your browser will accept a web site's cookies automatically when you visit the site. Such cookies are sent back to the web site on future visits.

- [● Enable all cookies]
- ○ Enable cookies for the originating web site only
- ○ Disable cookies
- [✓] Warn me before storing a cookie

Figure 8-2: Preferences in Netscape Navigator 6.x

4. You can select from the same options available in Netscape 4.x (although they are worded differently).

5. To view or delete stored cookies, click the View Stored Cookies button. This action will launch the Cookie Manager dialog box, shown is Figure 8-3.



View and Remove Cookies that are stored on your computer.

Site	Cookie Name
cnnaudience.com	AUDid
netscape.com	NS_REG
netscape.com	NS_WM
netscape.com	UIDC
yahoo.com	B

Information about the selected Cookie

```
Name:          UIDC
Information:   208.205.77.236:09708
Domain:        .netscape.com
Path:          /
Server Secure: No
Expires:       Fri Dec 31 16:00:16 2
```

☐ Don't allow removed cookies to be reaccepted later

Figure 8-3: Cookie Manager in Netscape Navigator 6

The Cookie Manager provides an interface for viewing, deleting and managing cookies in Netscape Navigator 6.0.

Microsoft Internet Explorer

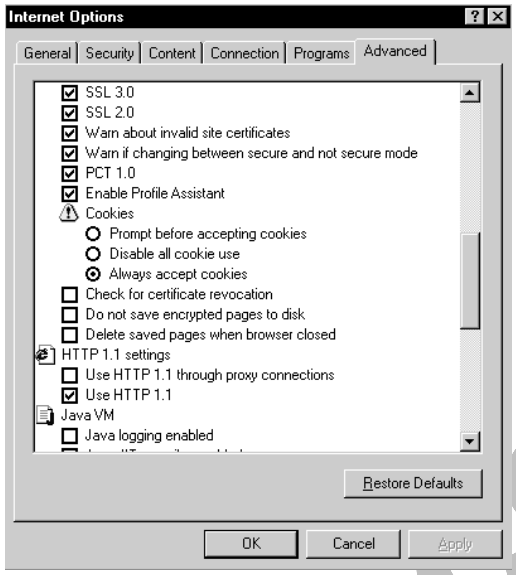
Follow these steps to enable or disable cookies in Microsoft Internet Explorer 4.x.

1. From the main menu, choose View.

2. Choose Internet Options from the View menu.

3. Click the Advanced tab to access cookie options.

4. Scroll down, then select the radio buttons shown in Figure 8-4.



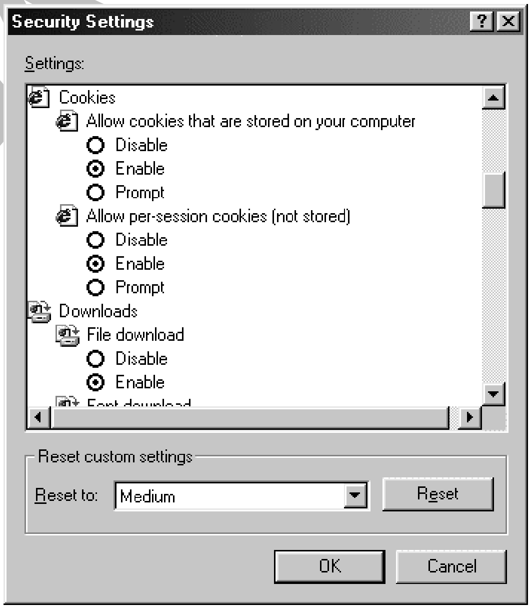
- [✓] SSL 3.0
- [✓] SSL 2.0
- [✓] Warn about invalid site certificates
- [✓] Warn if changing between secure and not secure mode
- [✓] PCT 1.0
- [✓] Enable Profile Assistant
- Cookies
  - ○ Prompt before accepting cookies
  - ○ Disable all cookie use
  - ● Always accept cookies
- [ ] Check for certificate revocation
- [ ] Do not save encrypted pages to disk
- [ ] Delete saved pages when browser closed
- HTTP 1.1 settings
  - [ ] Use HTTP 1.1 through proxy connections
  - [✓] Use HTTP 1.1
- Java VM
  - [ ] Java logging enabled

Figure 8-4: Enabling or disabling cookies in Internet Explorer 4.x

INSTRUCTOR NOTE: Make sure students know how to check the version of the browser in use. Both Microsoft Internet Explorer and Netscape Navigator provide a Help menu selection that provides version information for the browser.

Follow these steps to enable or disable cookies in Microsoft Internet Explorer 5.x.

1. From the main menu, choose Tools.
2. Choose Internet Options from the Tools menu.
3. Click the Security tab on the Internet Options dialog box.
4. Click the Custom Level button.
5. Scroll down in the Security Settings dialog box and select the radio buttons shown in Figure 8-5.



- Cookies
  - Allow cookies that are stored on your computer
    - ○ Disable
    - ● Enable
    - ○ Prompt
  - Allow per-session cookies (not stored)
    - ○ Disable
    - ● Enable
    - ○ Prompt
- Downloads
  - File download
    - ○ Disable
    - ● Enable

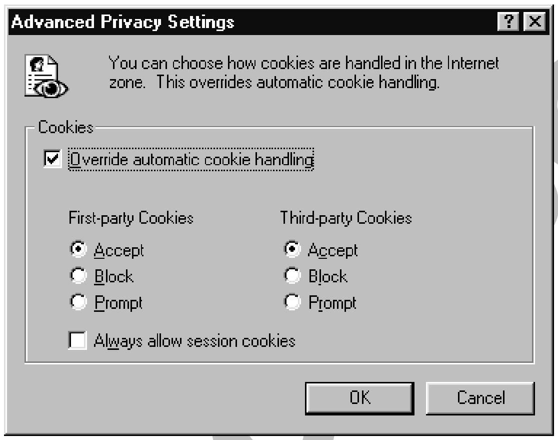
Figure 8-5: Enabling or disabling cookies in Internet Explorer 5.x

Note that Microsoft Internet Explorer 5.x allows you to differentiate between permanently stored cookies and per-session cookies.

Follow these steps to enable or disable cookies in Microsoft Internet Explorer 6.x.

INSTRUCTOR NOTE: By default, IE 6.x disallows third-party (or shared) cookies from sites that do not provide a privacy policy or that use personal information. Also, first-party cookies that use personal information are restricted if implicit consent is not provided by the user.

1. From the main menu, choose Tools.
2. Choose Internet Options from the Tools menu.
3. Click the Privacy tab on the Internet Options dialog box.
4. Click the Advanced button.
5. Select a cookie setting from the Advanced Privacy Settings dialog box, as shown in Figure 8-6.


Advanced Privacy Settings

You can choose how cookies are handled in the Internet zone. This overrides automatic cookie handling.

Cookies

[✓] Override automatic cookie handling

First-party Cookies	Third-party Cookies
[● Accept]	[● Accept]
Block	Block
Prompt	Prompt

[ ] Always allow session cookies

Figure 8-6: Enabling or disabling cookies in Internet Explorer 6.x

You will have an opportunity to create a cookie in the following lab.

Lab 8-1: Setting, viewing and clearing a cookie with JavaScript using Netscape Navigator

1. **System:** If necessary, start your Web server. Verify that your browser is set to warn you of incoming cookies.

2. **Editor:** Open the **lab8-1.htm** file from the Lesson_8 folder of the Student_Files directory.

3. **Editor:** Locate the existing <SCRIPT> tags in the <HEAD> section of the file. Locate the comment that reads as follows:

```
// Create cookie here
```

Add a line of code that creates a cookie named **`testCookie`**. Assign the variable **`myColor`** as its value. The following example shows the code before your changes.

```
<SCRIPT LANGUAGE="JavaScript">
<!--
var myColor = prompt("Please enter a color:","white");

if ((myColor == null) || (myColor == "") || ➘
   (myColor.length < 3)) x = "#ffffff";

// Create cookie here
```

```
alert (document.cookie);

var keepCookie = (confirm("Delete cookie?")) ? ➘
                  "delete" : "keep";

if (keepCookie == "delete") {
    document.cookie="testCookie=" + myColor + "; ➘
                    expires=01-01-97";
    message = "After deleting, your cookie looks like this: ";
    message += document.cookie;
    alert(message);
} else {
    message = "After keeping, your cookie looks like this: ";
    message += document.cookie;
    alert(message);
}
var pos = document.cookie.indexOf("=", 0) + 1;
var color = document.cookie.substring(pos, ➘
                                      pos + myColor.length);

//-->
</SCRIPT>
```

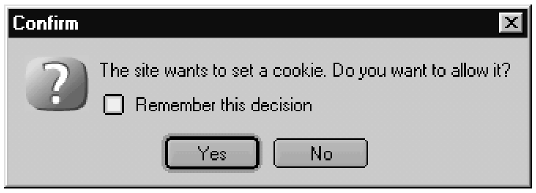
4. **Editor:** Examine the following code after the <BODY> tag in the file:

```
<SCRIPT LANGAUGE="JavaScript">
<!--
if ((color == null) || (color == "")) color="#ffffff";

document.bgColor = color;

document.write("Background color is: " + document.bgColor + "<BR>");
document.write("Cookie contents: " + document.cookie);

//-->
</SCRIPT>
```

5. **Editor:** Save **lab8-1.htm**.

6. **Browser:** Open **lab8-1.htm**. Setting the cookie will function properly with Netscape Navigator, or Microsoft Internet Explorer 4.0 or later. If you are using Internet Explorer 3.0, setting the cookie will function only if you retrieve the file through a server.

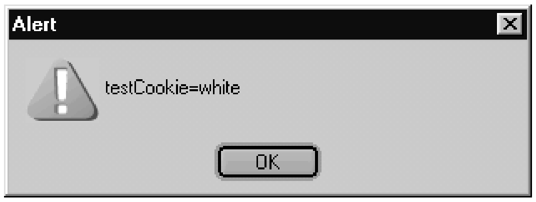
7. **Browser:** You should first see a prompt dialog box asking for a color. You should then see a series of questions and responses. You may see a message as shown in Figure 8-7.


Confirm

The site wants to set a cookie. Do you want to allow it?

Figure 8-7: Netscape Navigator cookie warning

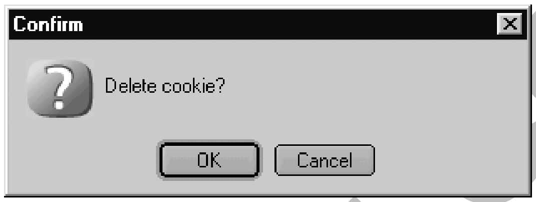
8. **Browser:** For this step, click **Yes** or **OK** to allow the cookie to be set. The next message, as shown in Figure 8-8, will specify the cookie you just accepted.


Alert

testCookie=white

Figure 8-8: Netscape Navigator cookie alert—cookie accepted

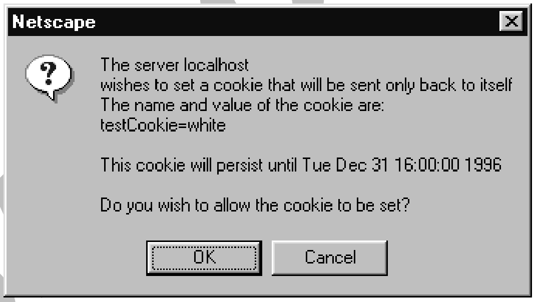
9. **Browser:** Click **OK** to continue. You will be asked whether to delete the cookie, as shown in Figure 8-9.


Confirm

Delete cookie?

Figure 8-9: Netscape Navigator delete cookie confirmation

10. **Browser:** Click **OK** to continue. If using Netscape Navigator 4.x, you will see the message warning you of the "new" cookie, which is really the same cookie with a backdated expiration date, as shown in Figure 8-10.


Netscape

The server localhost
wishes to set a cookie that will be sent only back to itself
The name and value of the cookie are:
testCookie=white

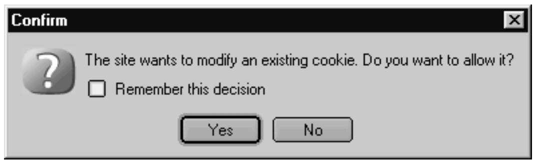
This cookie will persist until Tue Dec 31 16:00:00 1996

Do you wish to allow the cookie to be set?

Figure 8-10: Netscape Navigator 4.x cookie notice

11. **Browser:** If you are using Netscape Navigator 6.x, you will see a message warning you that a previously set cookie is being modified, as shown in Figure 8-11.

INSTRUCTOR NOTE: The student's cookie settings will determine whether the various warning dialog boxes will appear as shown in this lab.

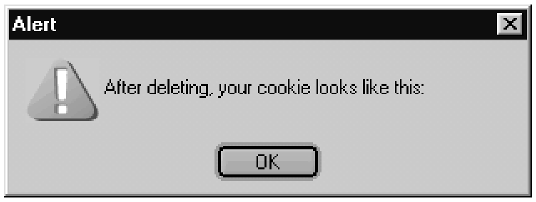

Confirm

The site wants to modify an existing cookie. Do you want to allow it?

☐ Remember this decision

Figure 8-11: Netscape Navigator 6.x cookie notice

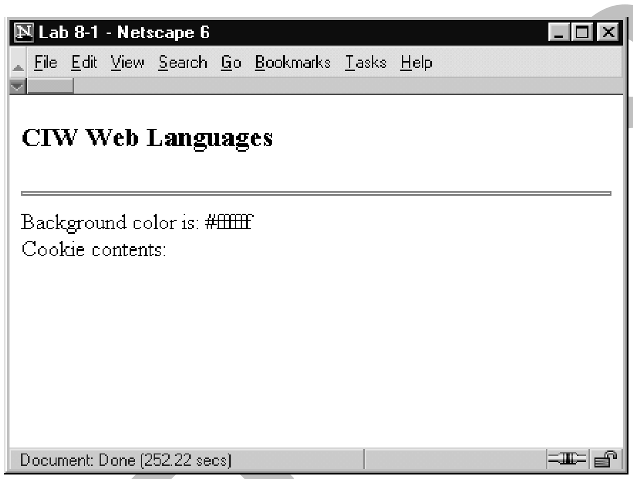
12. **Browser:** Click **Yes** or **OK** to set the expired cookie. This deletes the existing cookie with the same name. You will then see the alert shown in Figure 8-12.


Alert

After deleting, your cookie looks like this:

Figure 8-12: Netscape Navigator cookie alert—deleted cookie

13. **Browser:** Click **OK** to continue. The cookie is gone. Your screen should resemble Figure 8-13.


Lab 8-1 - Netscape 6

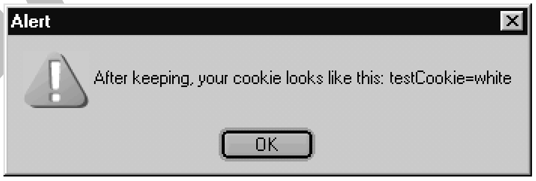
CIW Web Languages

Background color is: #ffffff
Cookie contents:

Document: Done (252.22 secs)

Figure 8-13: Lab8-1.htm

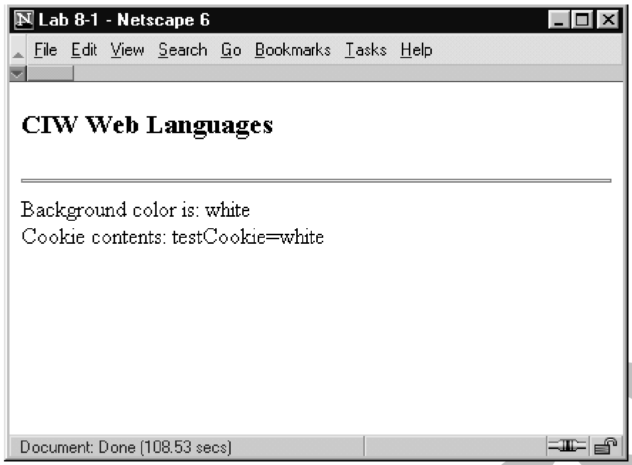
14. **Browser:** Reload the file. Repeat Steps 8 through 10, but this time, when asked if you want to delete the cookie, click **Cancel** or **No**. You will see the screen shown in Figure 8-14.


Alert

After keeping, your cookie looks like this: testCookie=white

Figure 8-14: Netscape Navigator alert after canceling deletion of cookie

15. Browser: Click **OK**. Your screen should resemble Figure 8-15.


Lab 8-1 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
Document: Done (108.53 secs)

CIW Web Languages

Background color is: white
Cookie contents: testCookie=white

Figure 8-15: Lab8-1.htm after canceling deletion

INSTRUCTOR NOTE: The code to create and extract cookie values can be somewhat complex. For interested students, a set of commonly used cookie functions written by Bill Dortch is discussed at *www.webreference.com/javascript/960701/part01.html*.

This lab demonstrates setting and retrieving cookie values. You learned how to set a cookie, delete a cookie, and apply the value from a cookie to the HTML document. Following is an examination of the first two lines of code from the this lab:

```
var myColor = prompt("Please enter a color:","white");
if ((myColor == null) || (myColor == "") || ➘
    (myColor.length < 3)) x = "#ffffff";
```

The first line declares the variable `myColor` and assigns it the value of the user's input captured by the prompt dialog box. The next line checks the variable `myColor` for various conditions, and assigns it a default value in case any of those conditions are true. You added the next line of code to create and assign a value to a cookie. After the line you added, the code is as follows:

```
alert (document.cookie);
var keepCookie = (confirm("Delete cookie?")) ? ➘
                 "delete" : "keep";
```

After the contents of the cookie are displayed to the user, the variable `keepCookie` is declared and assigned the result of a `confirm()` method. The value `delete` is assigned if the user chooses to delete the cookie (by clicking the OK button), or the value `keep` is assigned if the user chooses to keep the cookie (by clicking the Cancel button). Next, the code contains an `if...else` statement, as follows:

```
if (keepCookie == "delete") {
  document.cookie="testCookie=" + myColor + "; ➘
                 expires=01-Jan-97";
  message= "After deleting, your cookie looks like this: ";
  message += document.cookie;
  alert(message);
} else {
  message= "After keeping, your cookie looks like this: ";
  message += document.cookie;
  alert(message);
}
```

INSTRUCTOR NOTE:
The example code shown in this section demonstrates one way to extract the value from a cookie. Refer to **Instructor Note 8-1** for another demonstration of extracting the value from a JavaScript cookie.

If the variable `keepCookie` equals `delete`, the cookie named `testCookie` is assigned a backdated expiration date. The nonexistent cookie is then displayed to the user. If the variable `keepCookie` equals `keep`, the cookie is again displayed to the user. Next are the following lines of code:

```
var pos = document.cookie.indexOf("=", 0) + 1;
var color = document.cookie.substring(pos, ➘
                                      pos + myColor.length);
```

These lines of code extract the value portion of the cookie and assign that value to the variable `color`. The `indexOf()` method searches the cookie string for the equal sign (=), adds `1` to the return integer, and assigns that value to the variable `pos`. In the next line of code, the variable `pos` is used as the first argument of the `substring()` method. The value portion of the cookie string is being extracted, so the extraction should start one index position after the equal sign. The variable `pos` contains the integer that indicates this index position. The second argument, `pos + myColor.length`, determines the ending index position for the `substring()` method. For example, if the cookie's name/value pair is `testCookie=white`, the variable `pos` would be assigned the integer `11`. Recall that the variable `myColor` is assigned the user's color choice. Adding the variable `pos` together with the length of the string determines the ending index position for the `substring()` method. Using `testCookie=white` as the cookie's name/value pair, the second line in the preceding code would appear as follows upon execution:

```
var color = document.cookie.substring(11, 16);
```

The `substring()` method would extract the string starting at index position `11` and ending one character before index position `16`. Thus, the string `white` would be extracted.

Tech Note

*The lab8-1.htm program could have simply used the variable **`myColor`** as a value for the **`color`** variable. However, learning one method of extracting a value from a cookie name/value pair is beneficial because you will not always have access to the cookie's value as you did in the lab8-1.htm program.*

The following line of code is located in the body of the document:

```
if ((color == null) || (color == "")) color = "#ffffff";
```

This line assigns a value to the `color` variable in case the user has deleted the cookie. The remaining lines in the program assign the `color` variable as the value for `document.bgColor` and output the appropriate values to the screen.

Cookies and passwords

An optional lab for this lesson will demonstrate how to store a password in a cookie. Note that the functionality demonstrated by this lab should only be used for sites (or Web pages) that do not require strict security.

JavaScript Security Issues

Although JavaScript can write to the hard disk only in the form of cookies, it is still possible to manipulate JavaScript's inherent qualities to generate undesirable results. JavaScript can launch "helper" programs, which hackers can manipulate for their own purposes. JavaScript programs can include code that causes the browser to malfunction. Also, recent versions of JavaScript have introduced the concept of signed scripts. The following sections will briefly discuss these JavaScript issues.

JavaScript and helper application programs

Multipurpose Internet Mail Extensions (MIME)
E-mail protocol extensions that provide methods for encoding binary data for electronic transmission.

INSTRUCTOR NOTE: Stress to students that any damage caused by malicious JavaScript code would require the help of the user. The user would have to follow explicit instructions in order to cause any real damage with a helper application program.

If the user has defined helper applications for certain **Multipurpose Internet Mail Extensions (MIME)** types, a hacker can use JavaScript to launch them. This method is rather crude because it will also generate an "out of memory" error, which will alert an aware user that something is wrong. In addition, while the application will be launched, the launching of the application by itself is unlikely to cause damage without help from the user.

Plug-ins (programs that run within the browser) and helper applications (applications launched separately from the browser) rely on specified MIME information to define the file formats and extensions required by various plug-ins or helper applications. MIME types are defined by two pieces of information: the content type and the subtype. For example, a sound file might have a MIME type of audio/x-wav, wherein "audio" identifies the category or content type (a sound file) and "x-wav" indicates the subtype (WAV files). The "x" indicates that this file type is not a MIME standard type. Another MIME type might be application/msword, which would indicate a Microsoft Word file with a .doc extension.

Malicious and accidental coding

Accidental or malicious coding can generate infinite sequences that may cause the browser to malfunction. If this happens, the user must force the browser to quit. The following lab demonstrates this point.

Every programmer makes some mistakes while coding. However, ill-advised or malicious users could upload such a script to the World Wide Web deliberately. Other coding errors could cause similar problems. The goal of this section is to teach you to prevent—not cause—such errors.

Lab 8-2: Locking the browser with malicious code

In this lab, you will observe JavaScript used to lock a user's browser.

1. **Editor:** Open **lab8-2.htm** from the Lesson_8 folder of the Student_Files directory.

2. **Editor:** Examine the following source code:

```
<HTML>
<TITLE>Lab 8-2</TITLE>

<SCRIPT LANGUAGE="JavaScript">
<!--

for (i=0; i >= 0; i++) {
    alert("Stop me if you can!");
}

//-->
</SCRIPT>

</HEAD>

<BODY>
This page demonstrates malicious JavaScript code.
</BODY>
</HTML>
```

INSTRUCTOR NOTE: Ensure that students close all open programs before loading lab8-2.htm.

3. **Editor:** Close **lab8-2.htm**.

4. **System:** Close any open programs.

5. **Browser:** Open **lab8-2.htm**. You will see an alert dialog box as shown in Figure 8-16.

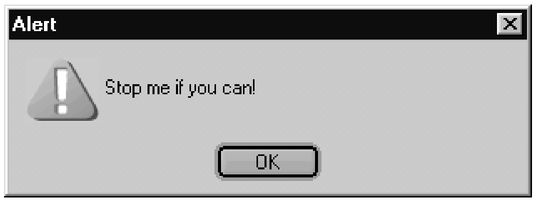
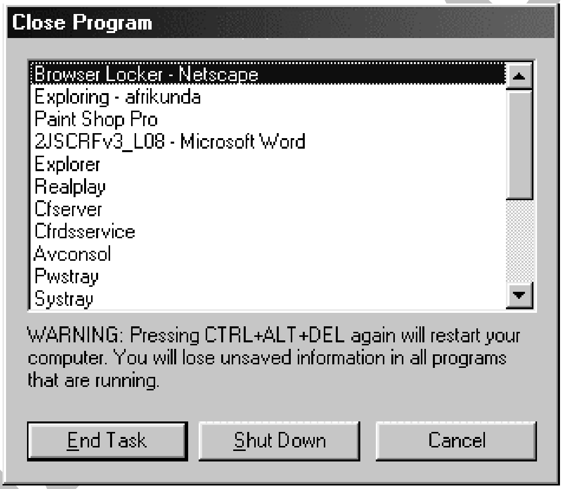


Figure 8-16: Alert dialog box

6. **Browser:** You can click the **OK** button repeatedly, but the message will return.

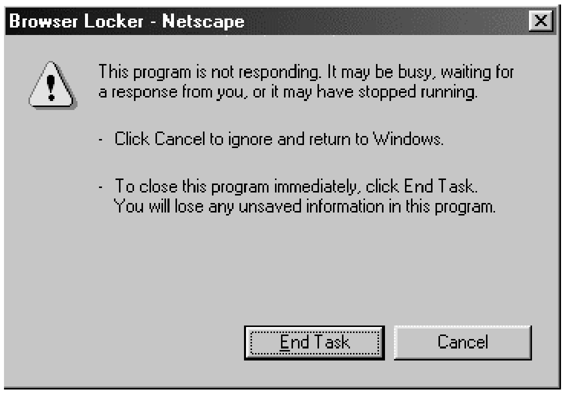
7. **System:** To stop execution of this script, you must close the browser. To do this, hold down the **CTRL** and **ALT** keys and press the **DELETE** key. This action will open a dialog box similar to Figure 8-17.



WARNING: Pressing CTRL+ALT+DEL again will restart your computer. You will lose unsaved information in all programs that are running.

Figure 8-17: Close Program menu

8. **System:** Verify that the Netscape option (or whichever browser you are using) is selected. Click the **End Task** button to close the browser. You will see the message shown in Figure 8-18.



This program is not responding. It may be busy, waiting for a response from you, or it may have stopped running.

- Click Cancel to ignore and return to Windows.

- To close this program immediately, click End Task. You will lose any unsaved information in this program.

Figure 8-18: End Task message

9. **System:** Click **End Task** again to complete the operation. You have now closed your browser.

Previous browser versions and security

You can create forms that capture and use a client's e-mail address when using Navigator 3.0. Usually, users have the option to be warned before this occurs. However, if this option is deactivated, JavaScript can be coded to send a message without a user's knowledge.

Note that although JavaScript helps mask the form and its purpose, mailing the contents of a form to yourself does not require JavaScript. It does, however, require the use of the Netscape browser.

The only way to attach data to a form and send it via e-mail using the Internet Explorer browser is to create a VBScript procedure. No single solution for mailing form information works on all browsers.

INSTRUCTOR NOTE: In order to create signed scripts, Netscape provides a script-signing tool. This tool is available at *developer.netscape.com/software/signedobj/jarpack.html*

Signed scripts

With the release of JavaScript 1.2, developers can create electronically signed scripts and Java applets. Signed scripts provide a mechanism by which programs can perform operations that were considered security risks in previous versions of JavaScript. An example of this type of operation is accessing a user's preference settings in the Netscape Navigator browser. Prior to the operation being executed, the user is always asked whether the operation should be allowed. The type of operation is clearly explained to the user, with access limited to specific areas of the client machine.

digital certificate
An electronic mechanism used to establish the identity of an individual or an organization.

The script signing process involves the use of **digital certificates**. A digital certificate is an electronic mechanism that establishes your identity and is issued by a certificate authority. The digital certificate is packaged in a Java Archive file (JAR file) and attached to the script using the `ARCHIVE` attribute of the `<SCRIPT>` tag. Netscape provides a JAR Packager tool to aid in the creation of signed scripts and applets.

The creation and use of signed scripts is beyond the scope of this course. As time allows, investigate the following URL for more information about signed scripts:

http://developer.netscape.com/docs/manuals/communicator/jssec/contents.htm

Lesson Summary

Application project

This project will challenge you to use some of what you have learned in this lesson.

Create an HTML page with a form. The form will contain a text box for the user's name and a submit button. Create a function in the `<HEAD>` section of the document that will set a cookie to the user's name submitted via the text box.

Add JavaScript code to the `<FORM>` tag. Use the `onSubmit` event handler to call the function that creates the cookie. Create code that informs the user if the name field is left blank. If the name field has a value, pass the user's name in the function call. In both the `if` clause and the `else` clause, use a `return false` statement to negate the submission of the form. The setting of the cookie is all that should occur for the `submit` event.

Add another button to the HTML form. Use an `onClick` event handler with this scriptable button to open a new window. Write code that will display the user's name (extracted from the cookie) in a welcome message in the document of the new window.

Skills review

Cookies are small memory-resident pieces of information sent from a server to a client's computer. A cookie is originally stored in the memory of the client's computer, but the cookie can be scripted to store this information on the hard drive. Cookies are sent from a server to the client's browser in the HTTP response header. Subsequent requests from the client can include the cookie information in the HTTP request header. Servers can then use that information for a variety of purposes, including storage of user preferences, user validation or maintenance of user state. JavaScript provides the `cookie` property of the `document` object to work with cookies. Any user can choose which cookies to accept by adjusting the settings in the browser. Some users may reject all cookies. Although JavaScript can write to the hard disk only in the form of cookies, it is still possible to manipulate JavaScript's inherent qualities to generate undesirable results. Malicious programs can launch helper applications or can contain code that will cause the browser to malfunction. Signed scripts provide a mechanism whereby programs can perform operations that were considered security risks in previous versions of JavaScript.

Now that you have completed this lesson, you should be able to:

- ✓ Explain cookies.
- ✓ Delete cookies from your disk.
- ✓ Assign a cookie.
- ✓ Test for the presence of a cookie.
- ✓ Clear a cookie.
- ✓ Enable and disable cookies in the browser.
- ✓ Use cookies and passwords to restrict entry to a page.
- ✓ Discuss security issues relevant to JavaScript.
- ✓ Define signed scripts.

Lesson 8 Review

1. What is a cookie?

   A cookie is a small piece of information sent from a server to a client's computer and stored in memory. It can be scripted to store its information more permanently on a user's hard drive.

2. What abilities does JavaScript offer the developer in relation to cookies?

   JavaScript allows developers to assign cookies, retrieve cookies and store cookies permanently on the hard drive.

3. How are cookies sent? What two actions concerning cookies can occur when a user generates an HTTP request?

   Cookies are sent from a server to a client's browser in the HTTP response header. When a user generates an HTTP request, any cookies already present that match the server's domain can be passed along in the request header, or a server can send a cookie back to the client in an HTTP response header.

4. How can cookies aid user authentication?

   A cookie can be used to store password information on a user's hard drive so he or she does not have to authenticate to access the site each time.

5. What types of information about a client can a cookie obtain?

   Some examples include operating system, browser type and version, service provider, IP address, a history of sites visited, and other types of information.

6. How can cookies aid state maintenance?

   A cookie can remember information about a particular connection and indicate where a user terminated activity in that connection. When the user returns, he or she can resume activity at the point of termination. Another example would be a cookie used to retain registration information so a return user need not re-enter his or her shipping address for a new order.

7. Which JavaScript object contains the `cookie` property that allows you to assign cookies?

   The `document` object.

8. How can you test for the presence of a cookie? What information is returned when you do this?

   One way to test for the presence of a cookie is by using an `alert()` method with the `document.cookie` statement in your script as follows:

   ```
   alert(document.cookie);
   ```

   When a user accesses the page, he or she will see a list of all cookies associated with that browser session. If no cookies are present, the user will see an empty alert.

9. Briefly describe how you can clear a cookie.

   Reassign the cookie, adding an expiration date that has already passed. The new cookie will replace the old cookie by the same name, and the cookie will immediately expire.

10. What capabilities do browsers provide for users to control cookies?

    Both Netscape Navigator and Microsoft Internet Explorer allow users to adjust settings so you can accept all cookies without warning, accept cookies only after receiving and confirming a warning, reject only shared cookies (Netscape only), or reject all cookies.

11. Briefly describe how you can use a password with a cookie. Is this a recommended method of security?

    You can store a password in a cookie so that JavaScript code can be used to allow access to a protected page only if the user enters the correct password. This method of security is not recommended for sensitive information; it is safer to store user names and passwords in a database and use server-side security features to authenticate users.

12. What are some security considerations related to JavaScript?

    JavaScript's inherent qualities can be manipulated to generate undesirable results. Hackers can use JavaScript programs to launch malicious helper applications. Also, accidental or malicious coding can generate infinite loops that can lock the browser and cause it to malfunction. Signed scripts can create programs that perform operations once considered to be security risks.

Lesson 8 Instructor Section

This section is a supplement containing additional tasks for students to complete in conjunction with the lesson. It also contains additional instructor notes. The instructor may use all, some or none of these additional tools, as appropriate to the specific learning environment. These elements are:

- **Additional Instructor Notes**
  Detailed instructor notes that expand the tips and suggestions presented in the instructor margin notes throughout the lesson.
- **Optional Labs**
  Computer-based labs to be completed during class or as homework.
- **Lesson Quiz**
  Multiple-choice test to assess student knowledge of lesson material.

Additional Instructor Notes

The following section contains detailed instructor notes that expand the tips and suggestions presented in the instructor margin notes throughout the lesson.

Instructor Note 8-1

Location: After Lab 8-1

This instructor note provides another example using JavaScript cookies. The following example demonstrates a cookie used for counting visits to an HTML page:

```
<SCRIPT LANGUAGE="JavaScript">
<!--

  var visits = parseInt(getCookie("numberOfVisits"));
  visits++;
  document.cookie = "numberOfVisits=" + visits ➘
   +  ";expires=Friday 26-Nov-2010 06:00:00 GMT";

  function getCookie(myCookie) {
    aCookie = document.cookie.split(";");
     for (var i = 0; i < aCookie.length; i++) {
       if (myCookie == aCookie[i].split("=")[0]);
          return aCookie[i].split("=")[1];
     }
     return 0;
  }

//-->
</SCRIPT>
</HEAD>
<BODY>

<SCRIPT LANGUAGE="JavaScript">
<!--

  document.write("You have visited this page <B>" + visits ➘
               + "</B> time(s).");

//-->
</SCRIPT>
```

In this example, the `String` object's `split()` method is used to extract the cookie's value. The `split()` method splits a string, placing individual pieces of the string into an array. Consider the following example:

```
var myNames = "Jerry, Mai, Juan";
var myNamesArray = myNames.split(",");

for (var i = 0; i < myNamesArray.length; i++) {
  document.write(myNamesArray[i] + "<BR>");
}
```

This example would output the names from the original string on separate lines of an HTML page. As shown in the previous example, the `split()` method takes the character to be used as a delimiter for the split operation as an argument. That character is not included in the resulting array.

In the earlier example, the `split()` method is first used to split the cookie using the semicolon as the delimiter. Inside the `for` loop, the array elements are again split using the equal sign as the delimiter. An `if` statement checks to see whether the name of the cookie matches the first array element, which holds the name of the cookie. Consider the following line of code:

```
return aCookie[i].split("=")[1];
```

This line of code returns the value of the `numberOfVisits` cookie, held as the second array element resulting from the `split()` method.

Optional Lab 8-1: Using passwords with cookies

In this lab, you will use a cookie that allows access to a second HTML page. If the cookie is present in the password-protected page and the user enters the correct password value, the user will be allowed to view the page. If the password is incorrect, the user will be returned to the previous page.

1. **Editor:** Open the **optionalLab8-1.htm** file from the Lesson_8 folder of the Student_Files directory.

2. **Editor:** Scroll down in the source code and examine the following code:

```
<FORM NAME="myForm">
<INPUT TYPE="text" NAME="pWord"><P>
<INPUT TYPE="button" VALUE="Submit" onClick="storePass(this.form);">
<INPUT TYPE="reset">
</FORM>
```

3. **Editor:** Locate the existing `<SCRIPT>` tags in the `<HEAD>` section of the document. Locate the comment that reads as follows:

```
// Create cookie here
```

   Create a cookie named `password` and assign as its value the user's entry in the `pWord` text box.

4. **Editor:** The following code shows the `storePass()` function before your changes:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
function storePass(form) {
   // Create cookie here


    alert(document.cookie);
    location.href = "pworded.htm";
}
//-->
</SCRIPT>
```

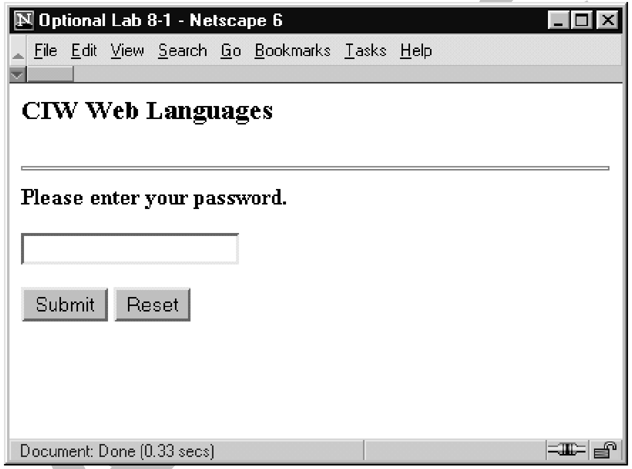
5. **Editor:** Save **optionalLab8-1.htm**.

6. **Editor:** Open **optionalLab8-1.1.htm** from the Lesson_8 folder of the Student_Files directory. Examine the following code in the <HEAD> section of the file:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
alert(document.cookie);
all = document.cookie;
if (all.indexOf("password=hello") != -1){
    alert("You have entered the correct password. Proceed.");
} else {
    alert("Incorrect password. Return and reenter.");
    location.href = "optionalLab8-1.htm";
}

//-->
</SCRIPT>
```

7. **Editor:** Close **optionalLab8-1.1.htm**.

8. **Browser:** Open **optionalLab8-1.htm**. Your screen should resemble Figure OL8-1.

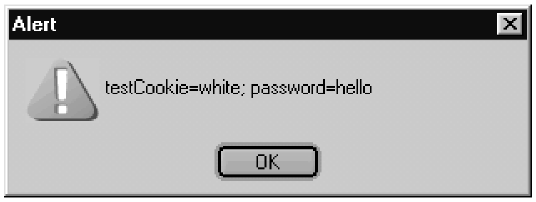

Optional Lab 8-1 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
Submit
Reset
Document: Done (0.33 secs)

CIW Web Languages

Please enter your password.

Figure OL8-1: OptionalLab8-1.htm

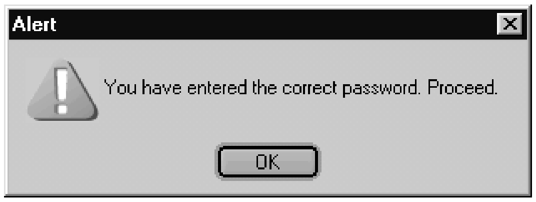
9. **Browser:** Enter the password **hello** and click the **Submit** button. This calls the `storePass()` function that stores your password into a cookie. Accept the cookie (if you are warned). You should then see the alert dialog box shown in Figure OL8-2.



testCookie=white; password=hello

Figure OL8-2: Alert dialog box

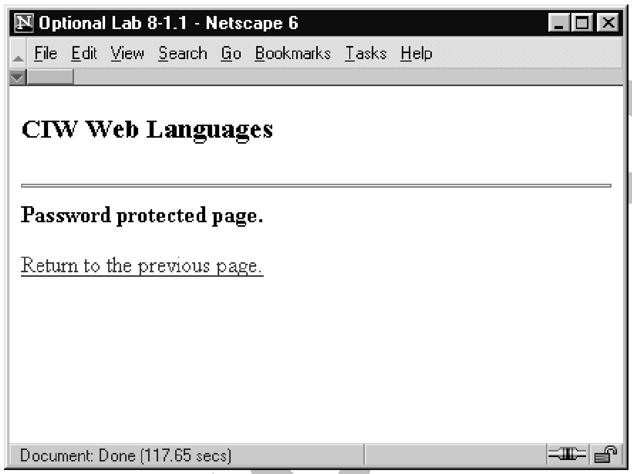
10. **Browser:** If you got a different message, you might have cleared the cookie assigned in the previous lab, in which case only the password *`name=value`* pair will appear. Click **OK** to continue. You should see another alert with the cookie's *`name=value`* pair, and then the alert dialog box shown in Figure OL8-3.


Alert

You have entered the correct password. Proceed.

Figure OL8-3: Alert dialog box

11. **Browser:** Click **OK** to continue. You will see the password-protected page, as shown in Figure OL8-4.


Optional Lab 8-1.1 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help

CIW Web Languages

Password protected page.

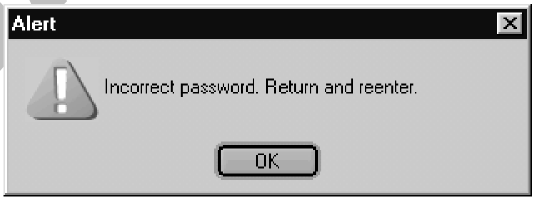
Return to the previous page.

Document: Done (117.65 secs)

Figure OL8-4: OptionalLab8-1.1.htm

12. **Browser:** To ensure that your script works for both correct and incorrect passwords, click the **Return** link to return to the previous page.

13. **Browser:** Enter an incorrect password and click the **Submit** button. After the alert displaying the cookie's `name=value` pair, you should see the message shown in Figure OL8-5.


Alert

Incorrect password. Return and reenter.

Figure OL8-5: Alert dialog box

14. **Browser:** Click **OK** to continue. Rather than seeing the file optionalLab8-1.1.htm, you should be returned to optionalLab8-1.htm.

This lab demonstrated another use of cookies. After you created the cookie named `password`, an alert displayed the cookie value to the user, and a `location.href` statement sent the user to the password-protected page. Following is the code from optionalLab8-1.1.htm:

```
alert(document.cookie);
all = document.cookie;
if (all.indexOf("password=hello") != -1){
    alert("You have entered the correct password. Proceed.");
} else {
    alert("Incorrect password. Return and reenter.");
    location.href = "optionalLab8-1.htm";
}
```

An alert displays the cookie's contents to the user. Next, a variable named `all` is assigned the value held in the cookie. An `if...else` statement then checks the string held in the variable `all` and sends the user back to password.htm if an incorrect password was entered. The most important line of code from this operation is the following:

```
if (all.indexOf("password=hello") != -1){
```

The `indexOf()` method is used to search the variable all for the string `password=hello`. If the integer returned by the `indexOf()` method is not `-1`, the cookie string must contain the string `password=hello`, meaning the user entered the correct password.

Note: This previous lab was presented for demonstration purposes only. Any sensitive content in a Web application should not be protected with the mechanism shown in this lab. It is much safer to store user names and passwords in a database. Server-side scripting in conjunction with the operating system's security features would then be used to authenticate and authorize users.

Lesson 8 Quiz

1. A cookie can be scripted to:

   a. *Track user activities on a server.*
   b. Obtain files from a user's hard drive.
   c. *Store information on a user's hard drive.*
   d. Obtain user information from a Web site.

2. Which cookie parameter is required?

   a. *`name=value`*
   b. `expires=date`
   c. `path=path`
   d. `secure`

3. Which example demonstrates the general JavaScript syntax to assign a cookie?

   a. `window.cookie = "name = value";`
   b. *`document.cookie = "name = value";`*
   c. `cookie = "expires = date";`
   d. `prompt(document.cookie);`

4. Which example demonstrates proper JavaScript syntax to test for the presence of a cookie?

   a. `document.cookie = "name = value";`
   b. `confirm(cookie);`
   c. *`alert(document.cookie);`*
   d. `window.cookie = "name = value";`

5. You have just reassigned a cookie with an expiration date that has already passed. What will occur?

   a. The cookie will expire and generate a same-named replacement.
   b. *The cookie will expire and be cleared.*
   c. The cookie will send a request for update to its server.
   d. Nothing. You cannot reassign a cookie in this way.

6. How can a user control cookies in his or her browser?

   a. *Adjust browser settings to choose which cookies to accept.*
   b. Adjust browser settings to avoid Web sites that set cookies.
   c. Use Netscape Navigator because it does not accept cookies.
   d. Use Microsoft Internet Explorer because it does not accept cookies.

7. What is a signed script?

   a. An electronic mechanism that establishes identity
   b. JavaScript code that has been copyrighted by the author
   c. JavaScript code that allows you to create a digital certificate
   d. *A program that can perform operations once considered a security risk*

8. How does a cookie appear in an HTTP response header?

   ```
   Set-Cookie: name=value; expires=date; path=path; domain=domain;
   secure
   ```

9. How would two cookies appear in a client's HTTP request header?

   ```
   Cookie: name=value; name2=value2
   ```

10. Write JavaScript code to create a function that sets a cookie named `user` that uses the user's input from the following form field as its value:

    ```
    <FORM NAME="myForm">
    ```

    Enter Name:

    ```
    <INPUT TYPE="text" NAME="name">
    <INPUT TYPE="button" VALUE="Set Cookie"
    onClick="setCookie(this.form);">
    </FORM>
    ```

    Set an expiration date of Saturday, December 31, 2005 at one second before midnight. Also, set the path to root (/) and indicate that a secure protocol should be used for the cookie.

    ```
    function setCookie(form) {
    var userName = form.name.value;
    document.cookie = "user=" + userName + "; path=/;expires=Saturday, ➘
    31-December-05 23:59:59 GMT; secure";
    ```

11. Consider the following malicious JavaScript code:

```
var myColors = new Array;
myColors[0] = "#FF0000";
myColors[1] = "#00FF00";
myColors[2] = "#0000FF";

for(var i = 0; i < myColors.length; i++){
  document.bgColor = myColors[i];
    if(i == 2){
      i = 0;
    }
}
```

Describe this malicious JavaScript code.

This malicious JavaScript code creates an array with three elements. The values for the elements are colors. Inside a `for` loop, the array's elements are assigned to the `bgColor` property of the `document` object. An `if` statement checks the value of the loop counter variable `i` and resets it to `0` if it equals 2. Thus, the user's screen constantly changes background colors for the HTML page, and the user must use the CTRL+ALT+DELETE mechanism to close the browser.

Lesson 9: Controlling Frames with JavaScript

Objectives

By the end of this lesson, you will be able to:

- Target frames with JavaScript.
- Change two or more frames simultaneously.
- Use functions and variables within framesets.
- Use functions and variables with related windows.
- Target the `opener` window.

Pre-Assessment Questions

1. Which of the following function invocations properly calls a parent window's function from a child window?

   a. `window.myFunction();`
   b. `parent.myFunction();`
   c. `top.myFunction();`
   d. *`opener.myFunction();`*

2. Which of the following code segments properly uses the `void()` syntax?

   a. *`<A HREF="javascript:void(childWin.addNums());">add numbers function</A>`*
   b. `<A HREF="void(childWin.addNums());">add numbers function</A>`
   c. `<A HREF="javascript://void(childWin.addNums());">add numbers function</A>`
   d. `<A HREF="void('childWin.addNums()');">add numbers function</A>`

3. Describe the functionality of the following JavaScript statement.

   ```
   top.frames[1].frames[1].location.href = "newPage.htm";
   ```

   This code changes the `location.href` property of the second frame defined in an embedded frameset. The first `frames[1]` refers to a second frameset defined in the `top` object using the `frames` array. The second `frames[1]` refers to the second frame defined in the embedded frameset.

Using JavaScript with Frames and Windows

Many users today have frames-capable browsers. Frames help to maximize the sophistication of your Web site by allowing you to make some information accessible as long as the site is open. This technique provides ease of navigation and a means to control the user's paths and destinations.

With JavaScript, you have complete, flexible control over the display, contents and coordination of frames on your site.

JavaScript also provides a means of communication between related windows. This lesson will conclude with examples of interaction between parent and child windows.

Understanding HTML frames and targets

INSTRUCTOR NOTE: The related appendix contains information concerning HTML frames. If your students are not familiar with frames, use the appendix to acquaint them with HTML frame functionality.

This lesson assumes that you understand the HTML `<FRAMESET>` tag and the syntax associated with targeting frames. A firm grasp of the HTML methods for targeting frames provides a foundation for understanding JavaScript's interaction with frames. If needed, refer to the appropriate appendix to review HTML frames and targets.

Targeting Frames with JavaScript

To target a frame in JavaScript, you need to know either its name or its number in the `frames` array. In addition, you must know its relationship in the frames hierarchy. As you know from your HTML experience, the keywords `_top` and `_parent` refer to frame relationships. In JavaScript, you omit the underscore characters. When you target `parent`, you are targeting the file containing the frameset in which the currently selected frame is defined. When you target `top`, you are targeting the parent of all parents.

In addition, frames in JavaScript are maintained in the `frames` array of the `window` object. Each frame is given an array index number starting from zero. The first frame in a frameset is `frames[0]`, the third frame is referred to as `frames[2]` and so forth.

You can target frames in either of two ways:

- By target name
- By the frame's number in the `frames` array

Examine the following simple frameset definition:

```
<!-- Filename: simpleFrame.htm -->
<FRAMESET COLS="25%, 75%">
     <FRAME SRC="a.htm" NAME="A">
    <FRAME SRC="b.htm" NAME="B">
</FRAMESET>
```

The frame named A is `frames[0]` and the frame named B is `frames[1]` in the `frames` array. To target frame B from frame A, the following syntax could be used:

```
parent.frames[1].location.href = "new.htm";
```

Alternatively, the following syntax could be used:

```
parent.B.location.href = "new.htm";
```

INSTRUCTOR NOTE: Refer to **Instructor Note 9-1** for information concerning frames and the **`top`** keyword.

To load a new document into the browser window that would replace the entire frameset, the following syntax could be used:

```
top.location.href = "new.htm";
```

In this situation, the keywords `parent` and `top` refer to the same entity. Thus, the following statement would produce the same results as the preceding example:

```
parent.location.href = "new.htm";
```

Obtaining a reference to a frame in a frameset such as simpleFrame.htm is straightforward. To obtain a reference to a frame in a second embedded frameset, more complex syntax must be used. Consider a frameset with the following structure:

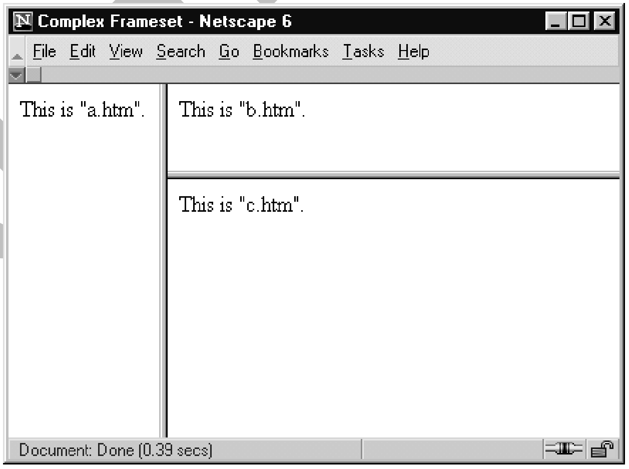
```
<!-- Filename: 3frames-1.htm -->
<FRAMESET COLS="25%, 75%">
    <FRAME SRC="a.htm" NAME="A">
   <FRAME SRC="3frames-2.htm" NAME="twofer">
</FRAMESET>
```

This frameset uses another embedded frameset. The second `<FRAME>` tag in the preceding code identifies the embedded frameset. The HTML code for the second frameset is as follows:

```
<!-- Filename: 3frames-2.htm -->
<FRAMESET ROWS="25%, 75%">
    <FRAME SRC="b.htm" NAME="B">
    <FRAME SRC="c.htm" NAME="C">
</FRAMESET>
```

If you open the file 3frames-1.htm in your browser, your screen should resemble Figure 9-1.

INSTRUCTOR NOTE: The **`parent`** property only applies to frames. In other words, it only applies to windows that are not top-level windows. The **`parent.parent`** syntax refers to the grandparent frame (or window) when a nested <FRAMESET> tag is defined within a child frame.


Complex Frameset - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
Document: Done (0.39 secs)

This is "a.htm".

This is "b.htm".

This is "c.htm".

Figure 9-1: 3frames-1.htm

In this situation, the term `parent` can have different connotations, depending on its usage. For instance, suppose the B frame contained the following link:

```
parent.frames[0].location.href = "new.htm";
```

This link would not change the first frame defined (the frame A) but would change the frame B, because the frame B is the first frame (`frames[0]`) defined in the embedded frameset. To access the frame A from the frame B, you would use the following syntax:

```
parent.parent.frames[0].location.href = "new.htm";
```

The parent.parent syntax is used to access the master frameset defined in the 3frames-1.htm file.

Though the frameset in the preceding figure could have been built without using an embedded frameset, conditions may make using embedded framesets compelling. You should know how to obtain references to frames when embedded framesets are used. The following examples demonstrate the various syntax options available for targeting frames with JavaScript.

To target frame B from frame A, the following syntax could be used:

```
parent.twofer.frames[0].location.href = "new.htm";
```

In the preceding example, the keyword parent returns a reference to the parent window object. The name of the embedded frameset, twofer, is used to return a reference to that frameset. Then, frames[0] returns a reference to the first frame defined in the embedded frameset. The following syntax could be used to achieve the same results:

```
parent.frames[1].B.location.href = "new.htm";
```

Again, the keyword parent returns a reference to the parent window object. Then, frames[1] returns a reference to the second defined frame in the frameset, which is the embedded frameset. The name of the desired frame, B, is used to reference the frame.

You can target frames in complex framesets using several options. Suppose you want to load a document into the frame A from the frame B or the frame C. Your options are as follows:

- `top.A.location.href = "new.htm";`
- `top.frames[0].location.href = "new.htm";`
- `parent.parent.A.location.href = "new.htm";`
- `parent.parent.frames[0].location.href = "new.htm";`

To target the B frame or the C frame from the A frame, you have several options (the examples target the B frame):

- `top.frames[1].B.location.href = "new.htm";`
- `top.twofer.B.location.href = "new.htm";`
- `parent.twofer.frames[0].location.href = "new.htm";`
- `parent.twofer.B.location.href = "new.htm";`
- `parent.frames[1].frames[0].location.href = "new.htm";`
- `parent.frames[1].B.location.href = "new.htm";`

In many cases, using the name of the frame for targeting purposes is the easiest method. Sometimes, you may not know the frame's name and must use the frames array to address the frame.

INSTRUCTOR NOTE: This section demonstrates how to use `button` objects and <A> tags to target frames. Students will apply this information in the next lab.

Two common mechanisms used to target frames are scriptable `button` objects and HTML <A> tags. The following is an example using a `button` object.

```
<FORM>
<INPUT TYPE="button" VALUE="new frame page" onClick=
"top.frames[1].myFrame.location.href='newPage.htm';">
</FORM>
```

In this example, an embedded frameset contains a frame named `myFrame`. The `frames[1]` refers to the second frameset defined in the top object. The `onClick` event handler is used to invoke the targeting code. The following is an example using an <A> tag for the same target.

```
<A HREF="javascript:void(top.frames[1].myFrame.location.href=
'newPage.htm');">new frame page</A>
```

This example demonstrates the use of the `javascript:` protocol to invoke the related code. Also note the use of the `void` keyword. Without the `void` keyword, the frame targeting code would still execute properly, but the page containing the link will also change URLs. In this example, the `void` keyword is used to prevent the link from functioning in its normal manner, whereas the `javascript:` protocol invokes the JavaScript code.

You will use a frames-enhanced version of a Web site in the next lab. The files needed are in the Lab_9-1 directory inside the Lesson_9 folder of the Student_Files directory. The file major.htm is the main frameset page. The file minor.htm contains a second, embedded frameset. The code for these files is as follows:

```
<-- Filename: major.htm -->
<HTML><HEAD><TITLE>Lab 9-1</TITLE>
</HEAD>
<FRAMESET COLS="30%,*">
   <FRAME SRC="toc.htm" NAME="toc">
   <FRAME SRC="minor.htm" NAME="twoframes">
<NOFRAMES>
<BODY>
If you had a frames-capable browser, you would see frames here.
</BODY>
</NOFRAMES>
</FRAMESET>
</HTML>

<-- Filename: minor.htm -->
<HTML><HEAD><TITLE>Lab 9-1 Second Frameset</TITLE>
</HEAD>
<FRAMESET ROWS="25%,*">
   <FRAME SRC="banner.htm" NAME="banner">
   <FRAME SRC="default.htm" NAME="main">
</FRAMESET>
</HTML>
```

Lab 9-1: Targeting frames

1. **Editor:** Open the **toc.htm** file in the Lab_9-1 folder (located in the Lesson_9 folder of the Student_Files directory). You will add script to several <A> tags that will load files into several targeted frames.

2. **Editor:** Add code to the <A> tags located in the <BODY> section of the document. Complete the HREF attributes by adding the appropriate syntax needed to target the various frames in the frameset. Try using the different types of syntax that were previously demonstrated in this lesson. The first link (to banner2.htm) should target the frame named `banner`. The next two links should target the frame named `main`. The CIW link should target the entire frameset.

3. **Editor:** The following code shows the <A> tags before your changes.

Note: The URL portions for the HREF statements have been provided for you.

```
<A HREF="             ='banner2.htm');">new banner</A><P>
<A HREF="             ='new1.htm');">new default page</A><P>
<A HREF="             ='default.htm');">original default page</A><P>
<A HREF="             ='http://ciwcertified.com');">CIW</A><P>
```

4. **Browser:** Open **major.htm**. Your screen should resemble Figure 9-2.

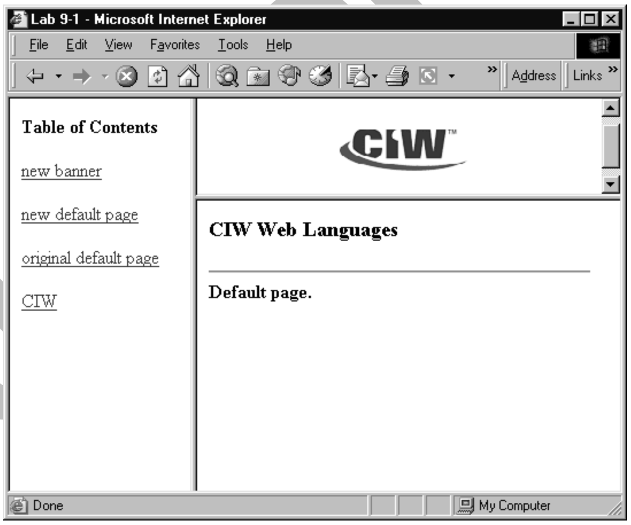

Lab 9-1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
CIW
Done
My Computer

Table of Contents

new banner

new default page

original default page

CIW

CIW Web Languages

Default page.

Figure 9-2: Major.htm

5. **Browser:** Click the **new banner** link. Your screen should resemble Figure 9-3.

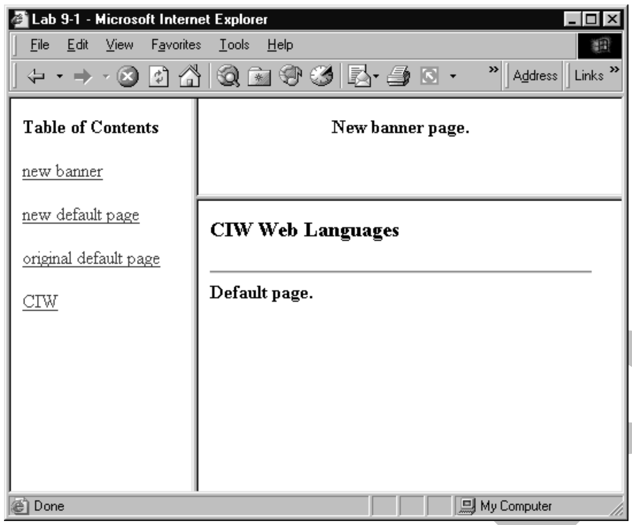

Lab 9-1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
Done
My Computer

Table of Contents

new banner

new default page

original default page

CIW

New banner page.

CIW Web Languages

Default page.

Figure 9-3: New banner loaded into banner frame

6. **Browser:** Click the **new default page** link. Your screen should resemble Figure 9-4.

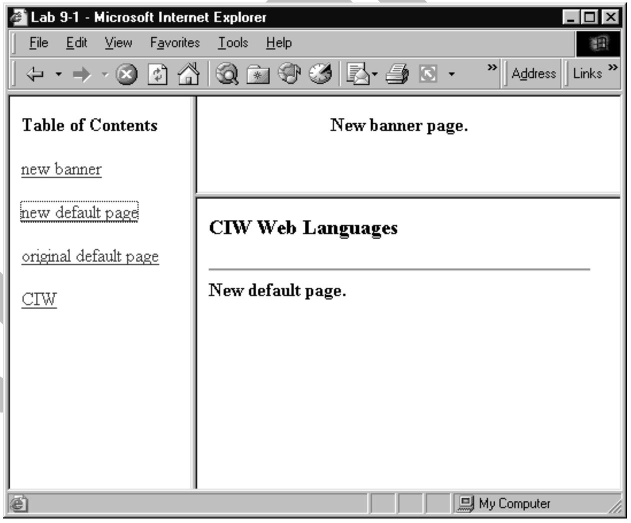

Lab 9-1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
My Computer

Table of Contents

new banner

new default page

original default page

CIW

New banner page.

CIW Web Languages

New default page.

Figure 9-4: New page loaded into main frame

7. **Browser:** Click the **CIW** link. Your screen should resemble Figure 9-5.

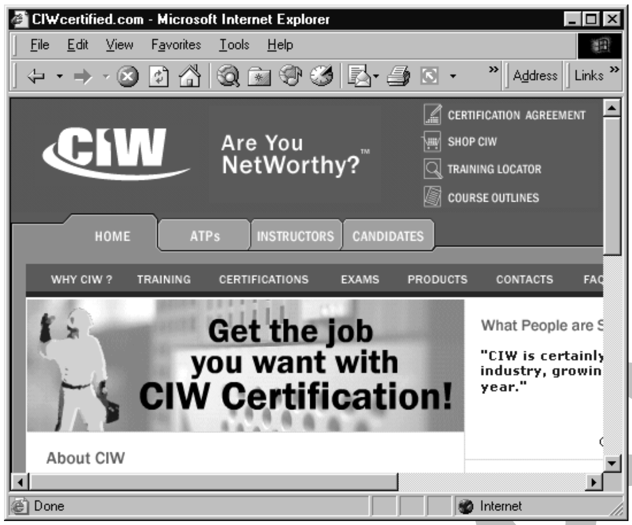


Figure 9-5: CIW Certified Web site

8. **Browser:** Click the **Back** button on the browser to return to the frames-enhanced Web site. Test all the links to ensure that they work properly.

In this lab, you learned how to target a frame with JavaScript for the purpose of changing the frame's contents. However, you might have other reasons to target a frame. For example, you may want to extract the title of the file currently appearing in any given frame, or you may want to change colors of the document in a particular frame or in a set of frames. To accomplish either of these tasks, the proper targeting procedure will be necessary.

Changing Two or More Frames with JavaScript

In the previous frames example, only one frame was changed per link. One way to change two frames simultaneously without using any script would be to target a frame with another file containing another frameset. This strategy would create the appearance of two frames changing simultaneously; however, only one file would actually change to another, with each file containing a frameset with two frames.

INSTRUCTOR NOTE: Refer to **Instructor Note 9-2** for another example using JavaScript to simultaneously change the documents in multiple frames.

Using JavaScript, you can change any combination of frames at any time. To change two or more frames at once in a script, write a simple function that includes two or more `location.href` statements. The following example creates a button that calls a function. The function changes the contents of the frames named `rightTop` and `rightBottom` in a frameset.

```
<SCRIPT LANGUAGE="JavaScript">
<!--

function changeFrames() {
    parent.rightTop.location.href = "newbanner.htm";
    parent.rightBottom.location.href = "newmain.htm";
}

// -->
</SCRIPT>
```

```
<FORM>
<INPUT TYPE="button" VALUE="Change Frames" onClick="changeFrames();">
</FORM>
```

You can change the contents of any frame or multiple frames, provided that you target the frame correctly using the frame name or `frames` array number in conjunction with the appropriate `parent` or `top` relationship.

Frames, Functions and Variables

INSTRUCTOR NOTE: This section demonstrates how to access functions and variables defined in parent frameset files. Students will apply this information in the next lab.

Functions and variables can be defined and stored in any of the files involved in creating a frameset. To reference a function or variable stored in a file other than the one from which the function or variable is being called, you need the appropriate reference. Two examples of the generic syntax follow:

```
relationship.frameName.variableName;

relationship.frameName.functionName();
```

In this syntax, *`relationship`* refers to either `parent` or `top`, and *`frameName`* refers to either the value of the NAME attribute in the <FRAME> tag or the `frames` array number representing the desired target frame.

For example, consider a variable that contains the user's background color preference, stored in the parent frame in a variable named `userBgColorChoice`. A child frame could reference this information by adding the following statement to a script block:

```
document.bgColor = parent.userBgColorChoice;
```

INSTRUCTOR NOTE: The parent document in a frameset provides an excellent central location for defining and storing global functions that may be needed throughout the entire frameset.

If you included a function named `testFunction()` defined in a parent frame, the child frame could call the function when you add the following line to a script block:

```
parent.testFunction();
```

In the next lab, you will call a function defined on a master frameset file from a child page. The calling statement will then receive the function's return value. You will also access a variable defined on a master frameset file from a child page.

Lab 9-2: Calling functions from parent and child frames with JavaScript

1. **Editor:** Open the **major.htm** file from the Lab_9-2 folder (located in the Lesson_9 folder of the Student_Files directory). Examine the following source code in the <HEAD> section of the document, then close the file:

```
<SCRIPT LANGUAGE="JavaScript">
<!–

var myParentVar = "The variable myParentVar is defined on ➘
                   the master frameset page.";

function myParentFunction(){
  alert("This function is defined on the master frameset page.");

  var myReturnValue = "This is the return value from ➘
  testFunction().";

  return alert(myReturnValue);
  }
```

```
//-->
</SCRIPT>
```

2. **Editor:** Open **toc.htm** from the Lab_9-2 folder. Modify the **`call function`** link. Use the `javascript:void()` syntax to invoke an `alert()` method that displays the value of `myParentVar`. Within the same <A> tag, use another `void()` statement to invoke the `myParentFunction()` function. Separate each `void()` statement with a semicolon.

3. **Editor:** Save **toc.htm**.

4. **Browser:** Open **major.htm**. Your screen should resemble Figure 9-6.

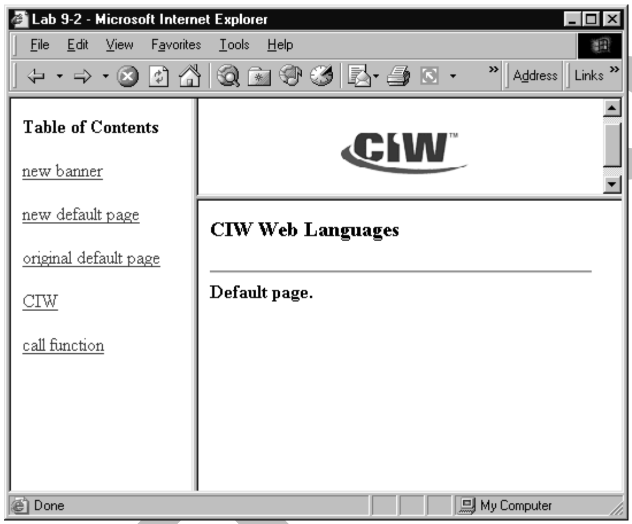
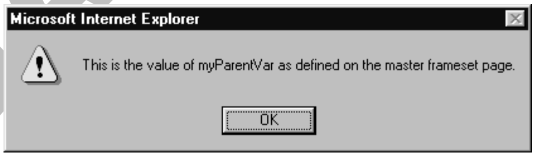


Figure 9-6: Major.htm

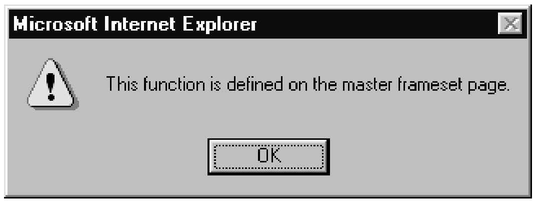
5. **Browser:** Click the **call function** link. You should see an alert dialog box as shown in Figure 9-7.



This is the value of myParentVar as defined on the master frameset page.

Figure 9-7: Alert dialog box with value of `myParentVar`

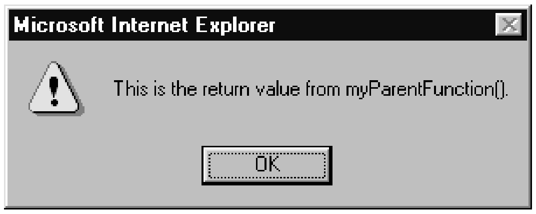
6. **Browser:** Click **OK**. You will see the alert dialog box defined in `myParentFunction()`, as shown in Figure 9-8.



This function is defined on the master frameset page.

Figure 9-8: Alert dialog box defined in `myParentFunction()`

7. **Browser:** Click **OK**. You will see the alert dialog box defined as the return value in `myParentFunction()`, as shown in Figure 9-9.



This is the return value from myParentFunction().

Figure 9-9: Alert dialog box defined as return value of `myParentFunction()`

INSTRUCTOR NOTE: Refer to **Instructor Note 9-3** for more information concerning communication between documents in a frameset.

8. **Editor:** If time allows, experiment with the function definition and function call for `myParentFunction()`. Pass a value from the calling statement as an argument for the function. Script `myParentFunction()` to use the value in some manner inside the function.

This lab demonstrates that two-way communication exists between parent frames and child frames. You were able to call a function defined on the parent frame page. The function was then executed, and it returned a value to the calling statement defined on the child frame page. A variable defined on the parent frame page was also accessed from the child frame.

Targeting Windows with JavaScript

With JavaScript, you can target documents in other windows as well. For example, consider the following line of code:

```
floatingWindow = open("","NewWin","width=200,height=375");
```

The variable `floatingWindow` returns a reference to the new window. You can use this reference to target the window as follows:

```
floatingWindow.location.href = "new.htm";
```

Targeting the *opener* window

When you open a new window, the parent window launches the newer window. You can refer to the parent window from the new window by referring to the `opener` property of the `window` object. The `opener` property can be used to access the methods and properties of the parent window.

For instance, from the new window, you can change the URL in the original window to display the CIW Web site by including the following in a script:

```
window.opener.location.href = "http://www.ciwcertified.com";
```

Windows, Functions and Variables

INSTRUCTOR NOTE:
Just as the parent document in a frameset provides an excellent central location for defining and storing global functions, parent windows provide a logical location for storing global functions and variables that may be needed by child windows.

The previously described relationship between parent and child frames also exists between parent and child windows. To access functions and variables defined on a parent window from the child window, you would include either of the following in a script:

```
window.opener.functionName();

window.opener.variableName;
```

To access functions and variables defined on a child window from the parent window, you would include either of the following in a script:

```
windowName.functionName();

windowName.variableName;
```

The following lab will demonstrate two-way communication between parent and child windows.

Lab 9-3: Calling functions from parent and child windows with JavaScript

1. **Editor:** Open the **lab9-3.htm** file from the Lesson_9 folder of the Student_Files directory. Locate the comment that reads as follows:

   `//Use opener property`

   The `myFunction()` function is scripted to open a new window and write content to that window's document. In the new window, a JavaScript function is defined. This function will be called from the parent window. The new window's document also includes a button scripted to call a JavaScript function defined on the parent page. Add an `onClick` event handler that invokes a function named `winFunction()`. Use the `window` object's `opener` property in the method invocation statement. The following shows the code before your changes:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
  function myFunction(){

  myWin = open("","","width=200,height=200");

    with(myWin.document){
      open();
      write("<HTML><HEAD><TITLE>Child Window</TITLE>");
     write("<SCRIPT>function myTest(){");
      write("alert('This function is defined in the ➘
                    child window");
     write(" and is called from the parent ➘
            window.'); this.focus()}");
      write("</SCRIPT></HEAD><BODY><H3>Child Window</H3><HR>");
     write("<FORM><INPUT TYPE='button' VALUE='Parent window ➘
      function' ");
      write("onClick='opener.winFunction();'>");
     write(<P><INPUT TYPE='button' VALUE='Close Window' ");

     // Use opener property

     write(                        );
      write("</FORM></BODY></HTML>");
```

```
        close();
      }
    }

    function winFunction(){
     alert("This function is defined in the parent window\n" + ➘
           "and is called by the child window.");
      myWin.focus();
    }

  //-->
  </SCRIPT>
```

2. **Editor:** Locate the comment that reads as follows:

   `<!-- Invoke child window function -->`

   Complete the <A> tag using the `javascript:void()` syntax to call the `myTest()` function defined in the child window. The child window is named myWin.

3. **Editor:** Save **lab9-3.htm**.

4. **Browser:** Open **lab9-3.htm**. Your screen should resemble Figure 9-10.

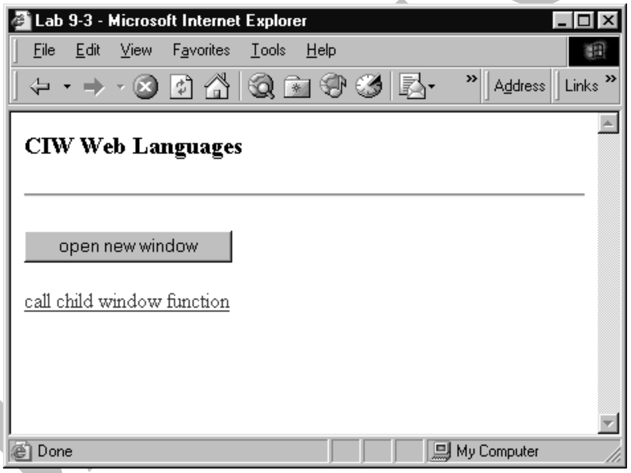

Lab 9-3 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
CIW Web Languages
open new window
call child window function
Done
My Computer

Figure 9-10: Lab9-3.htm

5. **Browser:** Click the **open new window** button. You will see a new window that resembles Figure 9-11.

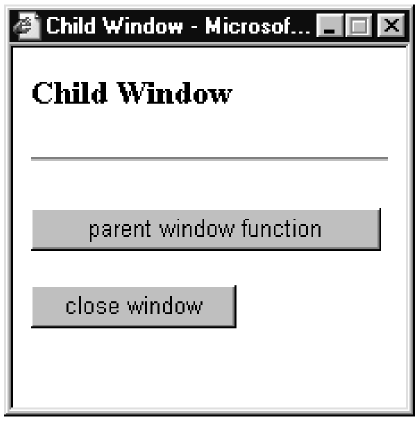
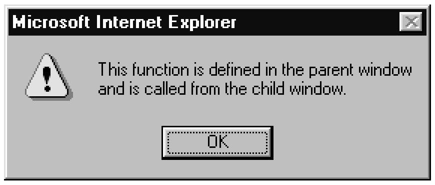


Figure 9-11: Child window

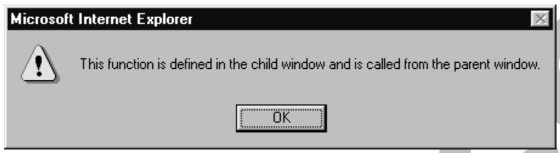
6. **Browser:** Click the **parent window function** button. You should see an alert dialog box as shown in Figure 9-12.



This function is defined in the parent window and is called from the child window.

Figure 9-12: Parent function called from child window

7. **Browser:** In the parent window, click the **call child window function** link. You should see an alert dialog box as shown in Figure 9-13.



This function is defined in the child window and is called from the parent window.

Figure 9-13: Child window function called

8. **Browser:** Click **OK**. Click the **close window** button to close the child window.

9. **Editor:** If time allows, experiment with the function definitions and function calls for `winFunction()` and `myTest()`. Pass values from the calling statements as argument for the functions. Script `winFunction()` and `myTest()` to use the values in some manner inside the functions.

This lab demonstrates that the two-way communication between parent and child frames also exists between parent and child windows. The lab again demonstrated syntax that can be used to launch functions from <A> tags. As mentioned previously, the `void` keyword is used to prevent a link from functioning normally, whereas the `javascript:` protocol invokes JavaScript code.

The functionality demonstrated in the previous lab can be critical to the operation of some Web applications. Learning how frames and windows communicate with each other is an important part of mastering the fundamentals of JavaScript.

INSTRUCTOR NOTE: See **Optional Lab 9-1: Building a frameset page with JavaScript**.

An optional lab for this lesson will provide an opportunity to use some of the skills you have learned thus far in this course.

Lesson Summary

Application project

This project will challenge you to use some of what you have learned in this lesson.

This application will feature an HTML page containing a form with a text box. The user will click a button from the same form that will launch a new window. The user will select a link from the new window. This action will invoke a function that writes information back to the text box in the parent window and will also close the new window. If the user does not click a link in the new window, a button will be scripted to close that window and at the same time write information back to the parent window indicating that no choices were made.

Templates for the form document and the document that will fill the new window have been created for you. They are named **classForm.htm** and **classList.htm**. These files are located in the Lesson_9_Application_Project folder in the Lesson_9 directory.

The classForm.htm file contains the HTML form. A button has been scripted to open a new window that displays the classList.htm file. The classList.htm file contains a list of CIW classes. Each class name is a link. Your task is to create a function that will be called when each link is clicked. You will also add code to the existing <A> tags to invoke the function. When the function is called, pass the class name of the related link as an argument. The function will write that text to the parent window's text box. The function will then close the child window.

Add a button to the classList.htm file. Script this button to write the text *no class chosen* to the parent window's text box. Then script the button to close the child window. The user can click this button if no link is selected.

Skills review

To target a frame in JavaScript, you need to know either its name or its number in the `frames` array. In addition, you must know its relationship in the frames hierarchy. Using JavaScript, you can change any combination of frames at any time. To change two or more frames in a script at a time, you write a simple function that includes two or more `location.href` statements. Functions and variables can be defined and stored in any of the files involved in creating a frameset, and can be accessed using the appropriate reference. Using JavaScript, you can target the properties and objects of parent and child windows. To refer to a parent window from a new window, use the `opener` property of the `window` object. To access a child window from the parent window, use the object reference obtained by assigning the `open()` method of the `window` object to a variable.

Now that you have completed this lesson, you should be able to:

- ✓ Target frames with JavaScript.
- ✓ Change two or more frames simultaneously.
- ✓ Use functions and variables within framesets.
- ✓ Use functions and variables with related windows.
- ✓ Target the `opener` window.

Lesson 9 Review

1. What is the advantage of using frames in a Web site? What benefits does JavaScript offer for using frames?

   Frames maximize the sophistication of a Web site by making some content accessible as long as the page is open, providing ease of navigation and a means to control the user's paths and destination. JavaScript gives you flexible control over display, contents, and communication between related frames and windows.

2. By what identity can you target a frame in JavaScript?

   By either its name (which is assigned by the `NAME` attribute in the `<FRAME>` tag) or its number in the `frames` array.

3. Name two JavaScript keywords you can use to target frames in JavaScript. Which frames do these keywords target?

   The `parent` keyword targets the file containing the frameset in which the currently selected frame is defined. The `top` keyword targets the parent of all parent files.

4. What is an embedded frameset? What syntax can be used to reference the master frameset in such a situation?

   An embedded frameset is a frameset within another frameset. The `parent.parent` syntax is one way to refer to the master frameset in this situation.

5. Generally, how can you change two or more frames simultaneously in JavaScript?

   Write a simple function that contains two or more `location.href` statements.

6. Can functions and variables be stored in frameset files? Can functions and variables be referenced by other files in the frameset? How?

   You can store functions and variables in any of the files involved in creating a frameset. You can access a function or variable in another frameset file with the appropriate reference.

7. Can you target the properties and objects of parent and child frames in a frameset? Describe an example.

   Yes, two-way communication exists between parent frames and child frames. For example, you can call a function defined on a parent page from a child page, execute the function and return a value to the calling statement defined on the child frame page.

8. When you open a new window, which window launches the newer window?

   The parent window launches the newer window.

9. Can you refer to a parent window from a new window that you have just opened? How?

   You can refer to the parent window from a new window by referring to the `opener` property of the `window` object. The `opener` property can be used to access the methods and properties of the parent window.

Lesson 9 Instructor Section

This section is a supplement containing additional tasks for students to complete in conjunction with the lesson. It also contains additional instructor notes. The instructor may use all, some or none of these additional tools, as appropriate to the specific learning environment. These elements are:

- **Additional Instructor Notes**
  Detailed instructor notes that expand the tips and suggestions presented in the instructor margin notes throughout the lesson.
- **Optional Labs**
  Computer-based labs to be completed during class or as homework.
- **Lesson Quiz**
  Multiple-choice test to assess student knowledge of lesson material.

Additional Instructor Notes

The following section contains detailed instructor notes that expand the tips and suggestions presented in the instructor margin notes throughout the lesson.

Instructor Note 9-1

Location: Targeting Frames with JavaScript

Students may ask how to programmatically prevent an HTML page from loading into a frameset. This task is accomplished by placing the following line of code in an HTML document's <HEAD> section.

```
if(window != top) top.location.href = location.href;
```

This code uses the != operator to determine if the `window` object for the current document is not equal to `top`. If it is not, the `location.href` property of the `top` object is set to the `location.href` property of the current document. In other words, the current document will occupy the topmost window, instead of any frame that may be part of the current window.

Students may also ask how to programmatically force an HTML page to be viewed only in a frameset. This task is accomplished by placing the following line of code in an HTML document's <HEAD> section.

```
// The url in the next line points to the main frameset page.
if (parent.location.href == self.location.href) ➘
window.location.href = "main.htm";
```

This code compares `parent.location.href` to `self.location.href`. If they match, the HTML page is attempting to occupy the topmost window. Also, the `window` object's `location.href` property is set to the HTML page that defines the frameset. Any page using this code cannot be viewed outside of its frameset.

Instructor Note 9-2

Location: Changing Two or More Frames with JavaScript

The following example demonstrates an alternative to using a JavaScript function to change two or more frames at one time.

```
<A HREF="javascript:void(parent.rightTop.location.href =
'newbanner.htm');void(parent.rightBottom.location.href =
'newmain.htm');">change two frames</A>
```

This example demonstrates that the `void` keyword can be used more than once within a single <A> tag. Note that a semicolon separates the individual JavaScript statements.

Instructor Note 9-3

Location: After Lab 9-2

Some security issues should be noted when referencing documents in other frames or windows. If the document happens to be from a different server, JavaScript will prevent access to that document's properties, as well as any functions or variables that may be defined therein. This restriction was implemented with the release of Netscape Navigator 2.02. Netscape Navigator 3.0 introduced data tainting, which allows JavaScript code to access properties of documents from different servers. With the release of JavaScript 1.2, signed scripts have replaced data tainting. If JavaScript code references a document in a frame or window that is from another server, a signed script must be used.

Optional Lab 9-1: Building a frameset page with JavaScript

In this optional lab, you will create a page that will allow users to check the price of CIW products and view an image of a product at the same time. This lab uses a simple frameset. The top frame will provide a drop-down menu from which the user can select a CIW product. You will create code that will write information to the bottom frame. The bottom frame will display an image of the product along with its name and price. Templates for this lab are provided in the OptionalLab_9-1 directory of the Lesson_9 folder in the Student_Files directory.

1. **Editor:** Open the **main.htm** file from the OptionalLab_9-1 folder. Examine the following code, then close the file:

```
<HTML>
<HEAD>
<TITLE>CIW Price Page</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
    function blankFrame() {
      return "<HTML><BODY> ➘
              <B>CIW items and prices</B></BODY></HTML>";
    }

    //-->
</SCRIPT>
</HEAD>
<FRAMESET ROWS="34%,*">
    <FRAME NAME="frame1" SRC="top.htm">
    <FRAME NAME="frame2" SRC="javascript:parent.blankFrame();">
</FRAMESET>
</HTML>
```

Note: This frameset uses a JavaScript function to supply a value for the SRC attribute of the second <FRAME> tag. The frame named `frame2` is the frame to which you will write.

2. **Editor:** Open **top.htm**. Locate the **`findPrice()`** function. The function begins with three variable declaration and initialization statements. Examine these statements:

```
var index = form.mySelect.selectedIndex;
var item = form.mySelect.options[index].text;
var price = form.mySelect.options[index].value;
```

INSTRUCTOR NOTE: The `blankFrame()` function is invoked using the `javascript:` protocol as discussed earlier in this lesson. The function returns HTML code used to fill the bottom frame when the page is initially opened. This mechanism demonstrates a convenient method for providing default content to a page that will be dynamically changed later by JavaScript code.

3. **Editor:** Following the variable statements, write code to open the data stream to the bottom frame. Then create a **`document.write`** statement to open `<HTML>`, `<BODY>` and `<H3>` tags. After the `<H3>` tag, concatenate the **`item`** variable into the string, then close the `<H3>` tag and add an `<HR>` tag. As shown previously, the `item` variable is assigned the `text` property from the `select` object.

4. **Editor:** Next, begin a **`switch`** statement using the `index` variable as the test expression. As shown previously, the `index` variable is assigned the integer returned from the `selectedIndex` property of the `select` object. For the `case 0` statement, add a **`break`** statement only. No action should occur if the user selects the default item from the drop-down menu.

5. **Editor:** For the `case 1` statement, add a **`document.write`** statement that will add an image to the bottom frame. For the `src` attribute, concatenate into the string the **`item`** variable (a corresponding .jpg file is found under each item's name in the images directory). Remember to add the images directory to the `src` attribute's value. Concatenate the .jpg portion of the image's URL and then a `<BR>` tag after the `item` variable.

6. **Editor:** Still inside the `case 1` statement, add another **`document.write`** statement that will output the item's price underneath the image. Concatenate the `price` variable into this string. The **`price`** variable is assigned the item's price (the value of the `VALUE` attribute in the `select` object's `<OPTION>` tags).

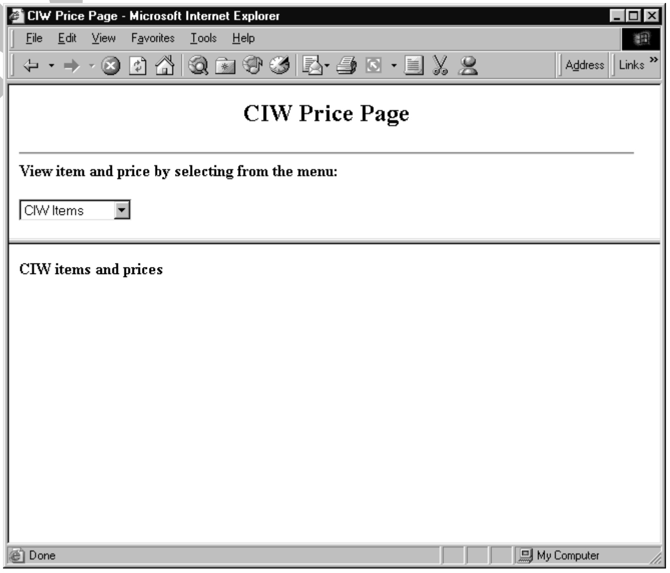
7. **Editor:** Include two more **`case`** statements so that the three CIW products can be viewed. Note that each `case` statement is identical. Only the integer value following the `case` keyword will change.

8. **Editor:** Close the **`switch`** statement.

9. **Editor:** Add a **`document.write`** statement to close the **`<BODY>`** and **`<HTML>`** tags for the bottom frame. Close the data stream to the bottom frame.

10. **Editor:** Examine the rest of the file, then save **top.htm**.

11. **Browser:** Open **main.htm**. Your screen should resemble Figure OL9-1.


CIW Price Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
CIW Items
Done
My Computer
CIW Price Page

View item and price by selecting from the menu:

CIW items and prices

Figure OL9-1: Main.htm

12. Browser: Select the first item from the drop-down menu. Your screen should resemble Figure OL9-2. If it does not, open **top.htm** in your editor and check your work.

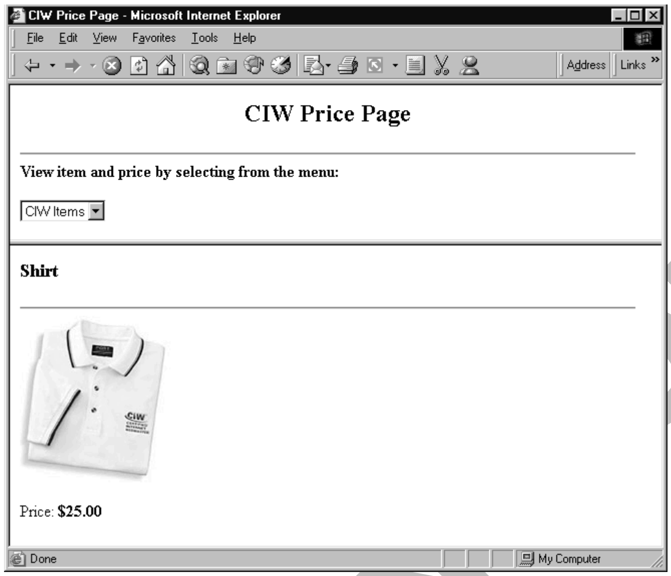

CIW Price Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
CIW Price Page
View item and price by selecting from the menu:
CIW Items
Shirt
Price: $25.00
Done
My Computer

Figure OL9-2: Main.htm with item selected

INSTRUCTOR NOTE: A completed version of this lab is available in the *Completed_Labs* folder on the supplemental CD-ROM.

13. Browser: Select the other two items that you have scripted. The appropriate information should be displayed in the bottom frame.

Lesson 9 Quiz

1. To target a frame in JavaScript, you need to know:

   a. Its parent frame reference.
   b. Its location in the embedded frameset.
   c. Both its name and its number in the `frames` array.
   d. *Either its name or its number in the `frames` array.*

2. Which JavaScript keyword targets the parent of all parent files in a frameset?

   a. `_top`
   b. `_parent`
   c. *`top`*
   d. `location`

3. To change two frames simultaneously, you must write a JavaScript function that includes what?

   a. The `parent.parent` statement
   b. Two `opener` statements
   c. Two references to `top`
   d. *Two `location.href` statements*

4. Which example shows the appropriate general reference syntax for accessing a function stored in a frame file?

   a. `frameName.functionName();`
   b. *`relationship.frameName.functionName();`*
   c. `relationship.frameName.variableName;`
   d. `frameName.opener.functionName;`

5. Which example shows the appropriate general reference syntax for accessing a variable defined on a parent window from the child window?

   a. `parent.windowName.variableName();`
   b. `child.frameName.functionName();`
   c. *`window.opener.variableName;`*
   d. `windowName. variableName;`

6. Which of the following allows you to access the methods and properties of the parent window from a newly opened window?

   a. The `location.href` statement
   b. The `open()` method of the `window` object
   c. *The `opener` property of the `window` object*
   d. The `parent.parent` statement

7. Which of the following allows you to access a child window from the parent window using an object reference?

   a. The `opener` property of the `window` object
   b. The `parent.parent` statement
   c. The `location.href` statement
   d. *The `open()` method of the `window` object*

8. A function named `myFunction()` is defined on a master frameset page. The frameset uses an embedded frameset defined in a file named 2Frame. The minor.htm file defines two frames named `bFrame` and `cFrame`. Following are the definitions of the frameset pages:

```
<!-- Main frameset -->
<HTML>
<HEAD>
<TITLE>Main Frameset</TITLE>
<SCRIPT LANGUAGE="JavaScript">
  function myFunction() {
    alert("myFunction() called.");
  }
</SCRIPT>
</HEAD>

<FRAMESET COLS="15%,*">
<FRAME NAME="toc" SRC="toc.htm">
<FRAME NAME="2Frame" SRC="minor.htm">
</FRAMESET>
</HTML>

<!-- Second frameset -->
<HTML>
<HEAD>
<TITLE>2Frame</TITLE>
</HEAD>

<FRAMESET rows="20%,*">
<FRAME NAME="bFrame" SRC="b.htm">
<FRAME NAME="cFrame" SRC="c.htm">
</FRAMESET>
</HTML>
```

How would you invoke `myFunction()` from the frame named `cFrame`?

Following are several ways to call the function:

```
parent.parent.myFunction();
top.parent.myFunction();
top.myFunction();
```

9. Consider the following JavaScript statement:

```
open("new.htm","newWin","height=300,width=300");
```

How would you change the background color of the document in the new window opened with this statement?

The new window has not been assigned an object reference, so you cannot change the background color of its document programmatically from the script that contains the preceding statement. Suppose an object reference had been used, such as the following:

```
myWin = open("new.htm","newWin","height=300,width=300");
```

_If so, then the following statement could access the document's `bgColor` ____ property:_

```
myWin.document.bgColor = "#0000FF";
```

10. Consider the following `myWin()` function:

```
var myWin;
function myFunction() {
  alert("myFunction() called.");
  myWin.focus();
}

function myWin(x) {
  myWin = open("","","height=200,width=200");

}
```

Complete the `myWin()` function. The function will open a data stream to the new window. Add a `button` object to the new window that will invoke `myFunction()` when clicked.

```
with (myWin.document) {
  open();
  write("<HTML><BODY><FORM>");
  write("<INPUT TYPE='button' VALUE='Click' ➘
onClick='opener.myFunction();'>");
  write("</FORM></BODY></HTML>");
  close();
}
```

Lesson 10: Custom JavaScript Objects

Objectives

By the end of this lesson, you will be able to:

- Create a custom JavaScript object.
- Define properties and methods of custom objects.
- Create new object instances.
- Create client-side databases using custom objects.
- Create functions and methods for manipulating client-side databases.

Pre-Assessment Questions

1. What is the term given to a JavaScript function used to create a custom object?

   a. customObjectConstructor
   b. *constructor*
   c. objectConstructor
   d. constructorFunction

2. Given that the `employeeObject()` function is properly defined, which of the following code snippets properly instantiates a custom JavaScript object?

   a. `var emp1 = employeeObject("Mai Lee", "Personnel", 2345);`
   b. `var emp1 = new function employeeObject("Mai Lee", ➘`
      `"Personnel", 2345);`
   c. `var emp1 = function employeeObject("Mai Lee", ➘`
      `"Personnel", 2345);`
   d. *`var emp1 = new employeeObject("Mai Lee", "Personnel", 2345);`*

3. Describe the process for creating a method for a custom JavaScript object.

   The method name for a custom JavaScript object is added to the special function used to create the custom object. For instance, if the custom object's method is named `findName()`, this line of code is added to the object's function: `this.findName = findName;`. The `findName()` method is then defined outside the object's function, like any other user-defined JavaScript function, using normal JavaScript function syntax.

Creating Custom Objects

INSTRUCTOR NOTE: Students are familiar with the concept of an array used to organize data. Before beginning this lesson, review the `Array` object. Remind students what an array is. Arrays can be constructed so that array elements appear as properties that can be accessed via dot notation. The `Array` object also provides methods that can operate on data structures in various ways. JavaScript custom objects take this concept a step further. Like `Array` objects, custom objects provide a convenient mechanism for creating data structures complete with their own properties and methods.

One of the most useful features of any object-based language is its ability to create objects. JavaScript provides useful predefined objects, such as `Date()`, `Array()` and `String()`. However, you are not limited to these objects. As a JavaScript programmer, you can create your own objects. However, you must understand how and when to create them.

Advantages of custom objects

Creating user-defined objects offers you two major advantages. First, you can create sophisticated solutions with minimal coding. Secondly, you can represent programming constructs as objects, allowing you to write efficient code schemes for evaluating forms or creating client-side databases.

Custom Object Demonstration

To further demonstrate the advantages of a user-defined object, you will define and deploy a JavaScript object in a client-side database. This project is in the form of a product information page for CIW products.

You will begin this project by creating a custom object named `productObject` that stores the product's ID, item name, description, price and image as properties. In addition, you will add a method to this object's definition that a client can then invoke for any product. Finally, you will display this information (which is held in instances of `productObject`) in various ways by using additional functions. As you follow this demonstration, you will learn the essential and most commonly invoked programming steps involved when creating a custom object.

Before moving on to the project, the next section will discuss the constructor, which is the building block of all custom objects.

Creating a JavaScript Object: The Constructor

For this lesson, an object is similar to a database record in that it can contain various fields or properties that are shared among all instances of that object. In addition to static data, you can construct objects to include certain methods and functions that enable you to control your object's behavior.

constructor
A special function that enables you to create instances of custom objects.

You define, or create, a custom JavaScript object with a special function called a **constructor**. The concept of a constructor in JavaScript is similar to that used in C++ and Java. The constructor defines the properties and methods of your object. Defining a constructor is the first step in creating an object.

Specifically, a constructor creates an empty template from which real-time objects, called instances, can be generated. The generic syntax for creating a constructor function is as follows:

```
function objectName(parameterA,parameterB,...) {
    this.property1 = parameterA;
    this.property2 = parameterB;
    this.method1 = functionName1;
    this.method2 = functionName2;
}
```

After you have examined the syntax that creates a constructor, insert some values. To create a product list, your newly constructed object will need properties to hold values for the following:

- Product ID
- Product item name
- Product description
- Product price
- Product image

You can call these parameters `id`, `item`, `description`, `price` and `image`.

INSTRUCTOR NOTE: Refer to **Instructor Note 10-1** for information concerning options available with constructor functions.

With only this information, you can code your constructor function as follows:

```
function productObject(id,item,description,price,image) {
  this.id = id;
  this.item = item;
  this.description = description;
  this.price = price;
  this.image = image;
}
```

This code defines a new JavaScript object template. Note that the object constructor is passed five parameters. These parameters will contain the object's properties when the object is created. Examine one statement from the constructor function:

```
this.description = description;
```

This statement can also be written as follows:

```
this.myDescription = description;
```

In other words, the two words do not need to match. However, the right side of the equation (in this case, the word `description`) must match the parameter that has been declared in the argument list for the function. In most cases, using the same name for the property on both sides of the equation is simplest.

INSTRUCTOR NOTE: The name of the constructor function should reflect the type of object being created. Using obscure names for constructor functions only makes the code harder to read and debug.

Note the use of the keyword `this` in the preceding example. As you learned previously, the keyword `this` is used to reference the object containing the script in which the keyword appears. In this case, after the constructor function is called, the keyword `this` returns the specific properties for the custom object to the calling statement, thus populating the new object with actual properties. This concept will become clearer when you actually create an instance of your custom object.

Once the constructor is defined, you need to create new instances of the object. This process is called instantiation. As you instantiate new copies of the object, you populate the object properties with information.

Creating an Instance of a Custom Object

INSTRUCTOR NOTE: This section demonstrates how to instantiate custom objects. Students may apply this information in **Optional Lab 10-1: Creating a custom object**.

To instantiate and then populate the properties of each new instance with actual data, you must declare variables. For example, if you want to add two products into your object model, you might include the following lines somewhere in a script block:

```
var backpack;
var hat;

backpack = new productObject("cp-01","CIW Backpack","Black woven backpack, blue-➘
green lettering","$25.00","images/backpack.jpg");

hat = new productObject("cp-02","CIW Hat","Brown canvas hat, ➘
blue-green lettering","$20.00","images/hat.jpg");
```

These lines create two instances of the `productObject` object, using `cp-01` as the `id` of the `productObject` named `backpack`, and so forth.

INSTRUCTOR NOTE: Refer to **Instructor Note 10-2** for another way to create and instantiate custom objects.

Usually you will want to use custom objects for repetitive tasks. For this reason, you will find it helpful to create your objects as elements of an array rather than creating individual variables as shown in the preceding example. To create an array, you need to declare a new instance of the `Array()` object. The following code shows how to store five `productObject` objects in the `products` array variable:

```
var products = new Array();

products[0] = new productObject("cp-01","CIW Backpack", ➘
Black woven backpack, blue-green lettering","$25.00", ➘
"images/backpack.jpg");

products[1] = new productObject("cp-02","CIW Hat","Brown canvas ➘
hat, blue-green lettering","$20.00","images/hat.jpg");

products[2] = new productObject("cp-03","CIW Suede Jacket", ➘
"Green suede, blue lettering","$110.00","images/suede.jpg");

products[3] = new productObject("cp-04","CIW Rain Jacket", ➘
"Green rainproof,blue lettering","$100.00","images/rain.jpg");

products[4] = new productObject("cp-05","CIW Polo Shirt", ➘
"White mesh polo,green lettering","$30.00","images/polo.jpg");
```

The preceding example creates, or instantiates, individual objects using the custom object definition found in the `productObject` constructor. After execution of the code, the new objects are in memory and available for manipulation as needed. Note the use of the keyword `new` in the instantiation statement. This concept is the same as using the keyword `new` in the `Array` object constructor.

INSTRUCTOR NOTE: Refer to **Instructor Note 10-3** for information on the **`prototype`** property and custom objects.

You use dot notation to access the data in the newly instantiated objects. Consider the following code:

```
document.write(products[0].id);
```

When executed, this code would output the `id` property of the first `products` element.

Creating Object Methods

To retrieve and view the information you just stored, you should define a method that allows you to view the data in instances of this `productObject` object. Again, you can create as many methods for your object as you need. You can make them as simple or as sophisticated as you like. For this project, you will create a method that allows a client to quickly open a new window and view the object information for an individual product. You will call the method `displayOne()`. You will then call this method from the Get Info button shown in Figure 10-1.

INSTRUCTOR NOTE: With older versions of JavaScript, all functions that acted as methods for a custom object had to be defined before the constructor function. If the constructor contained a reference to a function that was not yet defined, an error would be generated. This is no longer true. Using newer versions of JavaScript, the custom object's methods can be defined before or after the constructor function.

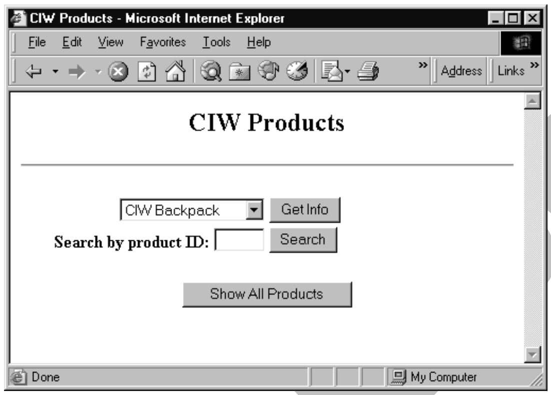


Figure 10-1: Product list page

To ensure that this method will work properly, you must first return to the constructor you just created and modify it to include the new method you are going to create. Remember that the constructor enables you to create `productObject` objects. For you to use the proposed method, you must first define it in the constructor. Note the line in bold in the following code:

```
  function productObject(id,item,description,price,image) {
    this.id = id;
    this.item = item;
    this.description = description;
    this.material = material;
    this.price = price;
this.displayOne = displayOne;
  }
```

INSTRUCTOR NOTE: Though the performance gains would be negligible in an application as small as this example, students can more efficiently create a variable and assign it the value of the **`products.length`** property. That variable would be used in the logical expression of the `for` loop. If you used this variable, the length of the array would not have to be evaluated each time through the loop. The code for the entire application at the end of the lesson will use such a variable.

Note that no parentheses follow the method name. If parentheses were placed after the method name, the code would call the function at that moment, which is not the intended action. When the JavaScript interpreter encounters this assignment statement, it searches the existing function definitions in the page. If a match is found, the JavaScript interpreter assigns the function to the identifier on the left side of the equation. The identifier is then automatically established as a method for the object.

The constructor's source code alone does not indicate that `price` is a property and `displayOne` is a method. You must create a separate function by the same name to make `displayOne` a method. However, before you create the method by means of a separate function, consider the HTML page on which you will use `productObject` objects. It is most helpful if you create the method in the context of the information on that page.

Ultimately, you will create a page that resembles the preceding figure. You will use JavaScript to populate the drop-down list using the `products` array. That portion of the file will resemble the following:

```
<SELECT NAME="itemName">

<SCRIPT LANGUAGE="JavaScript">
<!--

  for (var i = 0; i < products.length; i++) {
    document.write("<OPTION>" + products[i].item);
  }

//-->
</SCRIPT>

</SELECT>
```

The script block is located in the middle of the normal `<SELECT>` tag sequence. Instead of manually creating the individual options, you will use the `products` array and the `item` property to generate the text for each `<OPTION>` tag.

When an item is selected from the drop-down box and the button to its right is clicked, the method `displayOne()` will be executed. This method returns all the data stored in the properties of an individual `productObject` object to the screen.

You are almost ready to define the method `displayOne()`, which will be associated with your `productObject` object. If you study Figure 10-1 again, you will see the product selection drop-down box. The method you will create should display the item chosen in the drop-down box in a separate window.

First, examine the inline script that will invoke this method:

```
<INPUT TYPE="button" VALUE="Get Info"
 onClick="
 var i = document.prodForm.itemName.selectedIndex;
 products[i].displayOne();">
```

INSTRUCTOR NOTE: This section demonstrates how to extract data from a custom object. Students may apply this information in **Optional Lab 10-1: Creating a custom object**.

Remember that the select box was populated directly from the `products` array. Because both the `selectedIndex` property and the `products` array are zero-based, the number returned by the `selectedIndex` property will directly correspond to the appropriate `products` array index number. Furthermore, after the correct number has been returned, you will then invoke the `displayOne()` method on this single object.

Now you have the context for creating your `displayOne()` method. Examine the `displayOne()` function:

```
function displayOne() {
content = "";
content += "<HTML><HEAD><TITLE>" + this.item + "</TITLE></HEAD>";
content += "<BODY><DIV ALIGN='center'>";
content += "<TABLE WIDTH='90%'><TR><TD COLSPAN='2' ALIGN='center'>";
content += "<H3>Item: " + this.item.bold() + "</H3><HR>";
content += "<TR><TD><B>ID:</B> " + this.id;
content += "<TR><TD><B>Description:</B> " + this.description;
content += "<TR><TD><B>Price:</B> " + this.price.bold().fontcolor("#0000FF");
content += "<TR><TD><IMG SRC= '" + this.image + "' HEIGHT='75' WIDTH='75'>";
content += "</TABLE><FORM>";
content += "<INPUT TYPE='button' VALUE='OK' onClick='window.close();'>";
content += "</FORM></DIV></BODY></HTML>";
oneWindow = open("","OneWindow","width=450,height=290");
newWindow(oneWindow);
}
```

Note that the keyword `this` is used in the function to substitute property values for the object instance that called the `displayOne()` function. If the first item in the select list were chosen, you would call the method for `products[0]`. The keyword `this` would be replaced with `products[0]` upon execution. If a different item were chosen, a different array index number would be substituted.

INSTRUCTOR NOTE: The functions that are defined as methods for custom objects can receive arguments just as any normal user-defined JavaScript function can. These arguments (or variables that represent them) do not have to be defined in the object constructor function. They can simply be created and passed just like any other function parameter.

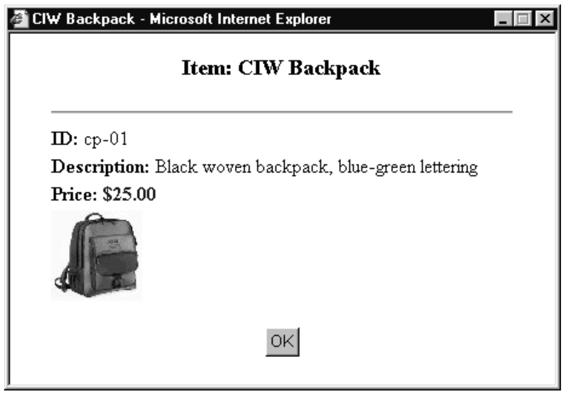
The last line in the function calls another function that opens a window and writes into it the pertinent content information created in the `displayOne()` function. Because you will need to write to a window from this function as well as from a later function, you want your window-writing function to be flexible and to write into the correct window. Each function will pass the name of the desired target window into the `newWindow()` function, where it will be stored and referred to as the variable `x`. Examine the following function, and consider its relation to the previous function:

```
function newWindow(x) {
    x.document.close();
    x.document.open();
    x.document.write(content);
    x.document.close();
   x.moveTo(20,20);
    x.focus();
}
```

The first line of the function, `x.document.close()`, is included to ensure a clean data stream for each `newWindow()` function call. The new window's `document` object is opened and the value of the `content` variable is written to the document. Note that the `content` variable is declared outside of any function and is global in scope. This declaration is appropriate because two different functions use this variable. The `window` object's `moveTo()` method is used to move the new window to the `x,y` coordinates of `20,20` (the `x` coordinate is 20 pixels from the left side of the screen and the `y` coordinate is 20 pixels down from the top of the screen). The last line of script will bring the window to the top of the window stack every time this function is called.

WARNING! *The* ***focus()*** *method, used in this context, will cause an error in Internet Explorer 3.0, which does not recognize the* ***focus()*** *method of the* ***window*** *object at this time. However, this method works well in Internet Explorer 4.0 and later, as well as Netscape Navigator 3.0 and later. Also, some older browsers may not support the* ***moveTo()*** *method. Remember that backward compatibility is always an issue when scripting for Web-based networking.*

Figure 10-2 shows the result of this method for the first item in the list: the CIW Backpack.


CIW Backpack - Microsoft Internet Explorer

Item: CIW Backpack

ID: cp-01
Description: Black woven backpack, blue-green lettering
Price: $25.00

Figure 10-2: After clicking Get Info

You can achieve the same effect without creating the `displayOne()` function as a method of the `productObject` object. However, if you do not create the function as a method, you cannot use the keyword `this` as shown, and the calling statement would be significantly different. Further, you would need to write a similar function to access the properties of each product. For this application, making the `displayOne()` function a method of the `productObject` makes good programming sense. Sometimes it is quicker to write the function as a method, but not always. The choice is yours, but remember that your approach should be appropriate to the situation.

In the next section, you will create two functions that are not methods, but are used to manipulate the product information in different ways.

Creating Functions for Your Objects

Methods operate on single instances of objects rather than all objects. When you need to evaluate multiple custom objects, in an array or otherwise, you need a function instead of a method. Functions that are not methods are not listed in the object constructor as the `displayOne()` method was. You will create two more functions to complete your product information page: a function that can search for an ID number and return the product's name, and a function that can list all products. You will call the search function `findItem()`, and you will name the function that shows all products `showAll()`.

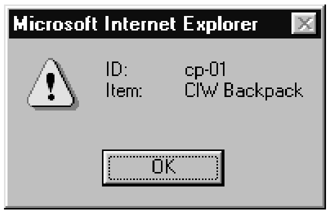
Examine the `findItem()` function:

```
function findItem() {
  var i = 0;

  while (i < products.length && products[i].id != ➘
  document.prodForm.idText.value.toLowerCase()) {
  i++;
  }

    if (i < products.length) {
      var message = "";
       message += "ID:\t" + products[i].id;
  message += "\nItem:\t" + products[i].item;
       alert(message);
     } else {
       var message = "It appears you have entered a ➘
       discontinued ID.\n";
       message += "Please try again.";
       alert(message);
    }
}
```

INSTRUCTOR NOTE: The code in the `findItem()` function again uses `products.length` in two different statements. Here is another example in which creating a variable to hold the value of `products.length` would be more efficient. As mentioned, the code for the entire application at the end of the lesson will use such a variable.

The `while` loop checks two conditions to determine the number of times that the loop will execute. As long as the loop counter variable `i` is less than the number of array elements and the current product ID does not equal the user-entered value, the loop will continue. Note that the only code inside the loop is the increment statement, `i++`. Thus when the loop ends, the variable `i` will contain a final value for the function. If the loop ends because the loop counter variable is no longer less than `products.length`, then all product IDs have been checked with no match found. If the loop ends because `products[i].id` equals the user's input, then a match has been found. The `if...else` statement checks the loop counter variable against `products.length`. If it is less than that value, the loop ended because a match was found. The code that shows the product information to the user will then execute. Note that the loop counter variable `i` is used to access the appropriate product data from the `products` array. If the loop counter variable is not less than `products.length`, a match has not been found, and the user is presented with that information.

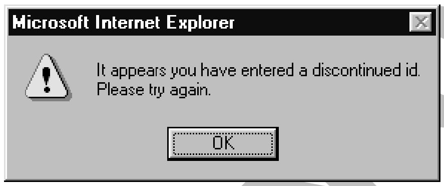
If you review your original product data at the beginning of this lesson, you will see that an item has the ID cp-01. If you were to enter this ID into the page and click the Search button, you would see the message shown in Figure 10-3.



ID: cp-01
Item: CIW Backpack

Figure 10-3: Search item for ID

Because the function includes code to process an invalid ID number, you should be able to enter either no text or text that does not represent an existing product ID, and receive the message shown in Figure 10-4.



It appears you have entered a discontinued id.
Please try again.

Figure 10-4: Search result for invalid entry

Now, create a `showAll()` function that opens a window that displays the names and prices of all products. Each product name will be a hyperlink. Clicking the hyperlink will invoke the previously created `displayOne()` method. When you present the user with data and your code generates links that lead to in-depth data displays, you allow the user to focus closely on the data a little at a time. The `showAll()` function will only provide the item name and price. When the user clicks the link to delve further into the data, he or she will be presented with the full data displayed by the `displayOne()` method.

Examine the `showAll()` function:

```
function showAll() {
content = "";
content += "<HTML><HEAD><TITLE>All CIW Products</TITLE></HEAD>";
content += "<BODY><DIV ALIGN='center'>";
content += "<TABLE WIDTH='85%' BORDER CELLPADDING='2'>";
content += "<TR><TH COLSPAN='2' ALIGN='center'>";
content += "CIW Product Information";
content += "<TR><TH>Item<TH>Price";

for (var i = 0; i < product.length; i++) {
  content += "<TR><TD>";
  content +="<A HREF='javascript:void(window.opener.products ➘
  [" + i + "].displayOne());'>";
  content += products[i].item + "</A>";
  content += "<TD>" + products[i].price;
}

content += "</TABLE><FORM>";
content += "<INPUT TYPE='button' VALUE='OK' onClick='window.close();'>";
content += "</FORM></DIV></BODY></HTML>";
fullWindow = open("","AllWindow", ➘
"width=450,height=385,resizable=1");
newWindow(fullWindow);
}
```

INSTRUCTOR NOTE:
Netscape Navigator does not display the source code for a dynamically created document in a new window. If the students want to see the result of concatenating the variable `i` into the `javascript:void` statement as discussed in this section, they should use Microsoft Internet Explorer or any browser that will display the source code for dynamically created documents.

Note that this code uses the `window.opener` property to find the JavaScript code necessary to execute. This property was discussed elsewhere in the course.

Like your `displayOne()` method, the last line of the function calls the `newWindow()` function to write content into a window. In this case, the content goes to the window named `fullWindow`.

As you can see, an <A> tag is created by the script. This tag creates a link to `products[i].displayOne()` in which `i` is the current `products` array index number. If you view the code in the window that pops up, the opening portion of the <A> tags will resemble the following:

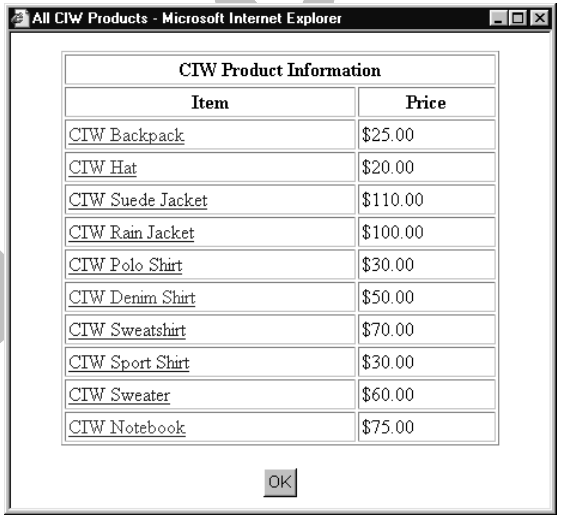
```
<A HREF=
'javascript:void(window.opener.products[3].displayOne());'>
```

Examine the piece of code that creates the opening portion of the links to the `displayOne()` method:

```
content+= "<A HREF='javascript:void(window.opener.products ➘
[" + i + "].displayOne());'>";
```

Note that the variable `i` is concatenated into the statement. Note also the use of a `javascript:void` statement. This statement uses the `javascript:` protocol to invoke the JavaScript code `window.opener.products[i].displayOne()`.

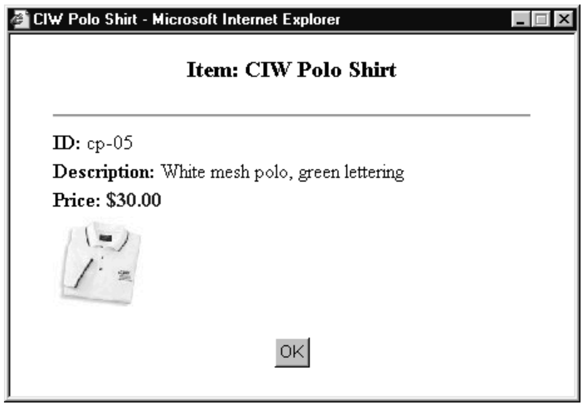
Now, observe the result when the Show All Products button is clicked. You should be able to see a new window with hyperlinks, as depicted in Figure 10-5.



CIW Product Information	
Item	**Price**
CIW Backpack	$25.00
CIW Hat	$20.00
CIW Suede Jacket	$110.00
CIW Rain Jacket	$100.00
CIW Polo Shirt	$30.00
CIW Denim Shirt	$50.00
CIW Sweatshirt	$70.00
CIW Sport Shirt	$30.00
CIW Sweater	$60.00
CIW Notebook	$75.00

Figure 10-5: Result of clicking Show All Products button

If you were then to click the CIW Polo Shirt link, a new window would appear, populated with the information shown in Figure 10-6.



Item: CIW Polo Shirt

ID: cp-05
Description: White mesh polo, green lettering
Price: $30.00

Figure 10-6: Result of selecting CIW Polo Shirt link

Now that you have seen all the individual parts of the client-side database, consider the full source code for the file named **CIWProducts.htm**.

INSTRUCTOR NOTE: The CIWProducts.htm file in the Completed_Labs folder contains code not shown in the coursebook. Specifically defined are two variables, named `oneWindowStatus` and `fullWindowStatus`, and a function named `closeAllWindows()`. These variables and the function are used to close the various windows opened by the application. Essentially, the `closeAllWindows()` function closes any dynamically created window whenever the main page gains focus. This prevents multiple dynamic windows from being open at one time.

```
<HTML>
<HEAD>
<TITLE>CIW Products</TITLE>

<SCRIPT LANGUAGE="JavaScript">
<!–
 var content, len;

 function productObject(id,item,description,price,image) {
   this.id = id;
   this.item = item;
   this.description = description;
   this.price = price;
    this.image = image;
   this.displayOne = displayOne;
 }

//Add products here
var products = new Array();
products[0] = new productObject("cp-01", "CIW Backpack", ➘
"Black woven backpack, blue-green lettering", "$25.00", ➘
"images/backpack.jpg");
products[1] = new productObject("cp-02", "CIW Hat", "Brown ➘
canvas hat, blue-green lettering", "$20.00", "images/hat.jpg");
products[2] = new productObject("cp-03", "CIW Suede Jacket", ➘
"Green suede, blue lettering", "$110.00", "images/suede.jpg");
products[3] = new productObject("cp-04", "CIW Rain Jacket", ➘
"Green rainproof, blue lettering", "$100.00", "images/rain.jpg");
products[4] = new productObject("cp-05", "CIW Polo Shirt", ➘
"White mesh polo, green lettering", "$30.00", "images/polo.jpg");
products[5] = new productObject("cp-06", "CIW Denim Shirt", ➘
"Dark blue denim, blue lettering", "$50.00", "images/denim.jpg");
products[6] = new productObject("cp-07", "CIW Sweatshirt", ➘
"Gray fleece pullover, green lettering", "$70.00", "images/sweat.jpg");
products[7] = new productObject("cp-08", "CIW Sport Shirt", ➘
"Blue, blue-green lettering", "$30.00", "images/sport.jpg");
products[8] = new productObject("cp-09", "CIW Sweater", ➘
"Blue scoop neck, short sleeve, green lettering", "$60.00", ➘
"images/sweater.jpg");
products[9] = new productObject("cp-10", "CIW Notebook", ➘
"Black leather, blue lettering", "$75.00", "images/notebook.jpg");

len = products.length;

function displayOne() {
  content="";
  content+="<HTML><HEAD><TITLE>" + this.item + "</TITLE></HEAD>";
  content+="<BODY><DIV ALIGN='center'>";
  content+="<TABLE WIDTH='90%'><TR><TD COLSPAN='2' ALIGN='center'>";
  content+="<H3>Item: " + this.item.bold() +  "</H3><HR>";
  content+="<TR><TD><B>ID:</B> " + this.id;
  content+="<TR><TD><B>Description:</B> " + this.description;
  content+="<TR><TD><B>Price:</B> " + this.price.bold().fontcolor("#0000FF");
  content+="<TR><TD><IMG SRC= '" + this.image + "' HEIGHT='75' WIDTH='75'>";
  content+="</TABLE><FORM>";
  content+="<INPUT TYPE='button' VALUE='OK' onClick='window.close();'>";
  content+="</FORM></DIV></BODY></HTML>";
  oneWindow = open("","OneWindow","width=450,height=290");
  newWindow(oneWindow);
}

function findItem() {
  var i = 0;
  while (i < len && products[i].id != ➘
document.prodForm.idText.value.toLowerCase()) {
    i++;
  }
  if (i < len) {
    var message = "";
     message += "ID:\t" + products[i].id;
     message += "\nItem:\t" + products[i].item;
     message += "\n\n";
     alert(message);
  } else {
    var message = "It appears you have entered a discontinued id.\n";
      message += "Please try again.";
      alert(message);
    }
}


function showAll() {
  content="";
  content+="<HTML><HEAD><TITLE>All CIW Products</TITLE></HEAD>";
  content+="<BODY><DIV ALIGN=center>";
  content+="<TABLE WIDTH='85%' BORDER CELLPADDING='2'>";
  content+="<TR><TH COLSPAN='2' ALIGN='center'>";
  content+="CIW Product Information";
  content+="<TR><TH>Item<TH>Price";

    for (var i = 0; i < len; i++) {
     content+="<TR><TD>";
      content+="<A HREF='javascript:void(window.opener.products[" + i + ➘
"].displayOne());'>";
      content+=products[i].item + "</A>";
      content+="<TD>" + products[i].price;
    }

  content+="</TABLE><FORM>";
  content+="<INPUT TYPE='button' VALUE='OK' î onClick='window.close();'>";
  content+="</FORM></DIV></BODY></HTML>";
  fullWindow = î open("","AllWindow","width=450,height=395,resizable=1");
  newWindow(fullWindow);
}

function newWindow(x) {
  x.document.close();
  x.document.open();
  x.document.write(content);
  x.document.close();
  x.moveTo(20,20);
  x.focus();
}

//-->
```

INSTRUCTOR NOTE: In the inventoryList.htm file, the variable `len` is assigned the value of `products.length` and is used in various statements throughout the program. As previously discussed, this is more efficient than continually evaluating `products.length` in various loops and other statements.

```
</SCRIPT>

</HEAD>
<BODY>
<DIV ALIGN="center">

<H2>CIW Products</H2>
<HR>
<FORM NAME="prodForm">
<TABLE WIDTH="100%">
<TR>
<TD WIDTH="50%" ALIGN="right">
<SELECT NAME="itemName">

<SCRIPT LANGUAGE="JavaScript">
<!--

  for (var i = 0; i < len; i++) {
    document.write("<OPTION>" + products[i].item);
  }

//-->
</SCRIPT>

</SELECT>
<TD>
<INPUT TYPE="button" VALUE="Get Info"
 onClick="
 var i = document.prodForm.itemName.selectedIndex;
 products[i].displayOne();">
<TR><TD ALIGN="right">
<B>Search by product ID:</B> <INPUT TYPE="text" SIZE="5" NAME="idText">
<TD>
<INPUT TYPE="button" value="Search" onClick="findItem();">
<TR><TD colspan="2" ALIGN="center">&nbsp;
<TR><TD colspan="2" ALIGN="center">
<INPUT TYPE="button" VALUE="Show All Products" onClick="showAll();">
</TABLE>
</FORM>
</DIV>
</BODY>
</HTML>
```

INSTRUCTOR NOTE: Custom JavaScript objects simulate database objects, but do not replace them. Custom JavaScript objects are effective when used with data that does not change often. They also provide a convenient alternative when the overhead of a database server or database application is not wanted or needed for an application. When used properly, JavaScript custom objects provide database-type functionality without a formal database server or database application.

As previously mentioned, no method is predefined for determining when you should use custom objects instead of arrays in your programs. An important question to ask about an object is: Can the object benefit from having a method or methods associated with it? As you gain experience with JavaScript, you will learn to view scripts that contain data structures in the context of custom objects. These JavaScript constructs are not the answer in every situation. When used in the appropriate circumstances, however, custom objects can add powerful functionality to your Web applications.

Complex Custom Objects

The `productObject` object defined in the preceding code is referred to as a simple custom object because only one level of properties and methods is associated with this object. JavaScript also supports complex objects. Complex objects are custom objects in which a property of the object is also an object complete with its own properties. See the related appendix for more information about complex custom objects.

Lesson Summary

Application project

This project will challenge you to use some of what you have learned in this lesson.

Choose an object with which you are familiar such as a bicycle or a car. Write down the properties that apply to this object. Create an HTML page that will display the properties of this object. In the context of this lesson, you will want to create a custom JavaScript object to represent the item. Use the steps outlined in this lesson to create and instantiate your custom object.

You will need to devise a way to display the information for your custom object. This task can be accomplished in many ways. Use concepts and skills that you have learned throughout this course. You may want to display information in alert dialog boxes, or you may want to create new windows to display the data. Experiment with various ways of displaying information for your custom object.

As you work on your project, experiment with creating individual custom objects and creating arrays of custom objects. With arrays of custom objects, experiment with the methods that are available for array manipulation. For instance, can you think of a use for the `Array` object's `sort()` method applied to an array of custom objects?

Skills review

As a JavaScript programmer, you can create your own custom objects. You define, or create, a custom JavaScript object with a special function called a constructor. The constructor defines the properties and methods of your object. Once the constructor is defined, new instances of the object are instantiated using the `new` keyword with the function name and a list of parameters. To retrieve the information associated with a custom object instance, you define a function that allows you to view the data. The function name is listed in the object constructor and thus becomes a method of the object. A method works on a single instance of an object. If needed, functions can be written to access the properties of multiple objects. Functions that are not methods are not listed in the object constructor. When used properly, custom objects can add powerful functionality to your Web applications.

Now that you have completed this lesson, you should be able to:

- ✓ Create a custom JavaScript object.
- ✓ Define properties and methods of custom objects.
- ✓ Create new object instances.
- ✓ Create client-side databases using custom objects.
- ✓ Create functions and methods for manipulating client-side databases.

Lesson 10 Review

1. Name two advantages of creating user-defined objects in JavaScript.

   You can create sophisticated solutions with a minimum of coding, and you can represent programming constructs as objects (which allows you to code efficient schemes for evaluating forms or creating client-side databases).

2. What special function allows you to create a custom JavaScript object? What two elements does this function define?

   A constructor. It defines the properties and methods of a custom object.

3. What is the term for the real-time objects that are generated from the empty constructor template?

   Instances.

4. What must you do after defining a constructor to implement its functionality? What is this process called?

   Once the constructor is defined, you need to create new instances of the object. This process is called instantiation.

5. How is the keyword `this` used with constructors?

   After the constructor function is called, the keyword `this` returns the specific properties for the custom object to the calling instantiation statement.

6. What must you do to instantiate and then populate the properties of each new instance with actual data?

   Declare variables that will become object references to the newly instantiated objects.

7. How can you create your object references to make them easier to use for repetitive tasks?

   Create your object references as elements of an array rather than creating individual variables.

8. How many methods can you create for an object?

   As many as you need.

9. When adding a method name to a custom object constructor, should you follow the method name with parentheses? Why or why not?

   No, you should not follow a method name added to a constructor with parentheses, because if parentheses are placed after the method name, the code would attempt to call the function at that moment. You want the JavaScript interpreter to encounter this assignment statement, assign the function name to the identifier, and automatically establish the method for the object.

10. With what do you operate on a single instance of a custom object? What should you use to evaluate multiple custom objects?

    A method operates on a single object rather than all objects. A function is used to evaluate multiple custom objects.

11. Does JavaScript offer a predefined method for determining when you should use custom objects instead of arrays for your programs? How else can you determine this?

    No, there is no predetermined method for this. Consider whether the object will benefit from having a method or methods associated with it.

12. What is a simple custom object? What is a complex custom object?

    A simple custom object has only one level of properties and methods associated with it. A complex custom object has a property or properties that are also objects complete with their own properties.

Lesson 10 Instructor Section

This section is a supplement containing additional tasks for students to complete in conjunction with the lesson. It also contains additional instructor notes. The instructor may use all, some or none of these additional tools, as appropriate to the specific learning environment. These elements are:

- **Additional Instructor Notes**
  Detailed instructor notes that expand the tips and suggestions presented in the instructor margin notes throughout the lesson.
- **Optional Labs**
  Computer-based labs to be completed during class or as homework.
- **Lesson Quiz**
  Multiple-choice test to assess student knowledge of lesson material.

Additional Instructor Notes

The following section contains detailed instructor notes that expand the tips and suggestions presented in the instructor margin notes throughout the lesson.

Instructor Note 10-1

Location: Creating a JavaScript Object: The Constructor

Starting with Netscape Navigator 4.x and Microsoft Internet Explorer 4.x, JavaScript provides a convenient mechanism for supplying default values for custom object properties. The following example demonstrates this concept.

```
function productObject(id,item,description,price,image) {
  this.id = id || "not supplied";
  this.item = item || "default item";
  this.description = description || "default description";
  this.price = price || "default price";
  this.image = image || "default image";
  this.displayOne = displayOne;
}
```

As demonstrated, the logical OR operator (||) is used to supply a value for each object property if a value is not supplied when the constructor function is invoked. If a value is passed in for a property, that value is assigned. Otherwise the value following the OR operator is assigned.

Instructor Note 10-2

Location: Creating an Instance of a Custom Object

Starting with Netscape Navigator 3.x and Microsoft Internet Explorer 4.x, JavaScript provides another mechanism for creating custom objects. This method uses the `new Object()` constructor. Consider the following example.

```
var backpack = new Object();
backback.id = "cp-01";
backpack.item = "CIW Backpack";
backpack.description = "Black woven backpack, blue-green lettering";
backpack.price = "$25.00";
backpack.image = "images/backpack.jpg";
backpack.displayOne = displayOne;
```

When you create more than one or two custom objects, the approach demonstrated in this lesson is more efficient. When you create single objects, the example shown in this instructor note is more efficient.

Starting with Netscape Navigator 4.x and Microsoft Internet Explorer 4.x, JavaScript provides yet another way to declare and instantiate custom objects. Consider the following example.

```
var backpack = {id:"cp-01", item:"CIW Backpack", ➘
description:"Black woven backpack, blue-green lettering", price:"$25.00", ➘
image:"images/backpack.jpg", ➘
displayOne:displayOne};
```

The curly braces (`{ }`) act as a call to the `new Object()` constructor. Object property name/value pairs are separated by a colon, as are method pointer/name pairs. The object's properties and methods are in a comma-delimited list. Complex custom objects can be created by inserting object or array references as properties of the object. Also, the variable used on the left side of the equation can be an array, making this construct a powerful approach to creating JavaScript data structures.

Instructor Note 10-3

Location: Creating an Instance of a Custom Object

As mentioned earlier in this course, the `prototype` property can be used to add new properties or methods to JavaScript objects. This property can also be used with custom JavaScript objects. Consider the following example.

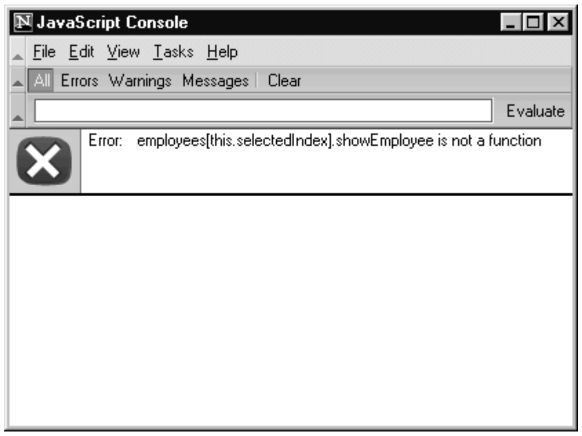
```
function productObject(id,item,description,price,image) {
  this.id = id;
  this.item = item;
  this.description = description;
  this.price = price;
  this.image = image;
  this.displayOne = displayOne;
}

var backpack = new productObject("cp-01", "CIW Backpack", ➘
"Black woven backpack, blue-green lettering", "$25.00", ➘
"images/backpack.jpg");

productObject.prototype.inStock = true;

alert(backpack.inStock);
```

When executed, this code would produce an alert dialog box containing the text *true*. The `inStock` property would be available to all `productObject` objects created for the document. The value for the `inStock` property could be overridden for any instance of the `productObject` object. Note that if any `productObject` instances did not have the value of the `inStock` property overridden, their `inStock` property would automatically inherit any value that might later be assigned to `prototype.inStock`.

Instructor Note 10-4

Location: Optional Lab 10-1

When the code for optionalLab10-1.htm is executed in the Netscape Navigator 6.2 browser, the browser returns an error message as shown in Figure IN10-1.



Error: employees[this.selectedIndex].showEmployee is not a function

Figure IN10-1: Error message in Netscape Navigator 6.2

Specifically, this error message is generated if the user selects the *Select Employee* option from the drop-down menu. The `select` object is scripted with the `onChange` event handler to execute the appropriate showEmployee() function when an employee name is selected. In theory, nothing should happen if the first option is selected because no change has occurred (the change event has not been executed). This appears to be a bug in Netscape Navigator 6.2. Note that the error message does not appear in the browser window but in the JavaScript console. Also, this error message does not prevent the rest of the application from functioning properly. The optionalLab10-1.htm file executes with no error messages in Netscape Navigator 4.7. Also, no error messages are reported with Microsoft Internet Explorer 6.0.

The file optionalLab10-1.1.htm provides a workaround for this situation. The following is an excerpt from that file.

```
<SELECT NAME="empName" onChange="
             if(this.selectedIndex == 0) {
               return;
             }
             else {
                employees[this.selectedIndex].showEmployee();
                this.selectedIndex=0;
             }">
```

In this workaround, the value of the selectedIndex property is evaluated before the call to the showEmployee() function. If the value is 0, the return keyword stops execution of the code. This code prevents the error message from being generated in Netscape Navigator 6.2.

Optional Lab 10-1: Creating a custom object

INSTRUCTOR NOTE: Refer to **Instructor Note 10-4** for information concerning the *optionalLab 10-1.1.htm* file included in the Lesson_10 folder of the Completed_Labs directory.

This lab will provide an opportunity for you to create and instantiate a custom object. You will then write code to display data pertaining to the custom object.

1. **Editor:** Open the **optionalLab10-1.htm** file from the Lesson_10 folder of the Student_Files directory.

2. **Editor:** Locate the comment that reads as follows:

   `// Complete the employeeObject constructor`

3. **Editor:** Complete the employeeObject constructor function that has been started for you. Add the properties that you see listed in the employeeObject constructor signature. Also add a method named showEmployee.

4. **Editor:** Locate the comment that reads as follows:

   `// Instantiate 3 instances of employeeObject`

5. **Editor:** Instantiate three instances of the employeeObject. Use the employees array that has been declared for you. Note that the first element of the employees array has already been defined. Make sure you start your array index numbers with 1 for the first employee. The values for the employeeObject properties have been provided for you.

6. **Editor:** Locate the comment that reads as follows:

   `// Complete the showEmployee() function`

7. **Editor:** Complete the `showEmployees()` function. Use the `info` variable that has been declared for you. Use the += operator to build a string containing the text **Employee:**, **Department:** and **Extension:** with the appropriate `employeeObject` data concatenated in the appropriate locations. Concatenate a line break character after the name and department information.

8. **Editor:** An `alert()` method has been defined in the `showEmployees()` function that will display the `info` variable.

9. **Editor:** Examine the rest of the code, then save **optionalLab10-1.htm**.

10. **Browser:** Open **optionalLab10-1.htm**. Your screen should resemble Figure OL10-1.

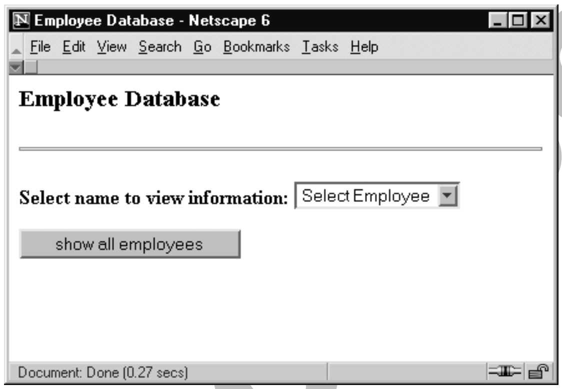
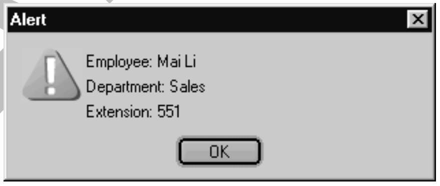

Employee Database - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
Employee Database
Select name to view information:
Select Employee
show all employees
Document: Done (0.27 secs)

Figure OL10-1: OptionalLab10-1.htm

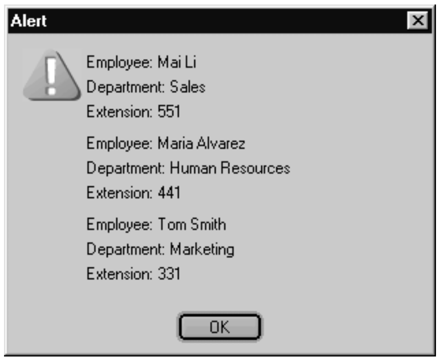
11. **Browser:** Select the first name from the drop-down menu. You should see an alert dialog box as shown in Figure OL10-2. If you do not, check the code you entered in optionalLab10-1.htm.



Employee: Mai Li
Department: Sales
Extension: 551

Figure OL10-2: Alert displaying employee information

12. **Browser:** Click the **show all employees** button. You should see an alert dialog box as shown in Figure OL10-3.



Employee: Mai Li
Department: Sales
Extension: 551

Employee: Maria Alvarez
Department: Human Resources
Extension: 441

Employee: Tom Smith
Department: Marketing
Extension: 331

Figure OL10-3: Alert showing all employees

13. **Browser:** Test all the names in the drop-down menu. Ensure that the proper information is displayed for each employee. If the proper data is not displayed, check the code you entered in optionalLab10-1.htm.

This lab offered you hands-on experience in creating, instantiating and displaying the data for a JavaScript custom object. You added code to a constructor function to make it a template for your custom objects. You added code to instantiate instances of your custom object, populating the object's properties with actual values. You also added code that extracted and displayed the data from your custom object.

This code in this lab used the `onChange` event handler to execute the `showEmployees()` function. Consider the following code.

```
<SELECT NAME="empName" onChange="employees[this.selectedIndex].showEmployee();
this.selectedIndex=0;">
```

As shown, the `selectedIndex` property of the `select` object is used as the subscript for the `employees` array. This determines the instance of the `employeeObject` that will invoke the `showEmployee()` function. As the value of the first element of the `employees` array is the text *Select Employee*, nothing should happen when that option is selected from the drop-down menu. The `selectedIndex` property of the `select` object is set back to `0` after each user selection. This action is in anticipation of the next user selection, and also ensures that the `onChange` event handler will never execute code if the first option is selected.

Though this lab created only three custom employee objects, hundreds of instances of the `employeeObject` could be instantiated, with little performance degradation. Remember that custom objects can provide a powerful client-side alternative to database servers or database applications when used in the right situations.

Lesson 10 Quiz

1. What is the first step in creating a custom object?

   a. Creating a method
   b. *Defining a constructor*
   c. Naming property values
   d. Creating a property

2. What is a constructor?

   a. A new instance of a custom object
   b. A method that defines the properties of a custom object
   c. A predefined object to which you add methods and properties
   d. *A function that defines methods and properties of a custom object used as a template for instances of the custom object*

3. What is instantiation?

   a. The process of defining a constructor
   b. *The process of creating new copies of an object*
   c. The process of defining properties and methods of a custom object
   d. The process of returning specific method parameters to the calling statement

4. How do you instantiate new instances of an object?

   a. *With the new keyword, the function name, and a list of values*
   b. With the `new` keyword and the object name
   c. With the `this` keyword and the object name
   d. With the `this` keyword, the function name, and a list of parameters

5. How do you access the properties and methods of a newly instantiated object?

   a. With the `this` keyword
   b. By declaring variables
   c. *Using dot notation*
   d. With the `new` keyword

6. Which of the following operates on single instances of custom objects rather than on all objects?

   a. *Methods*
   b. Functions
   c. Properties
   d. Arrays

7. Define a constructor function for a custom object named `empObject`. Add three properties: `name`, `age`, and `department`. Add one method: `showOne`

```
function empObject(name, age, department) {
   this.name = name;
   this.age = age;
   this.department = department;
   this.showOne = showOne;
   }
```

8. Using the `empObject()` constructor function from the previous question, instantiate two new employees in an array named `employees`.

```
var employees = new Array();
employees[0] = new empObject("Jane Doe", 25, "Management");
employees[1] = new empObject("Jose Ruiz", 35, "Sales");
```

9. Using the first employee instantiated in the previous question, how would a program access the `name` and `age` properties for that object? Output the data to two separate lines on an HTML page.

```
document.write("Name: " + employees[0].name + "<BR>");
   document.write("Age: " + employees[0].age + "<BR>");
```

10. Using the `employees` array, write a script block that would output each employee's name while making each name a link to the `showOne()` method. You will define the `showOne()` method in the next question.

    One way to write this code is as follows:

```
var x = employees.length;
for (var j = 0; j < x; j++) {
document.write("<A HREF='javascript:void(employees[" + j  + ➘
"].showOne())'>");
document.write(employees[j].name + "</A><BR>");
}
```

11. Consider the following `showOne()` function:

```
function showOne() {
}
```

Complete the `showOne()` function so that an alert dialog box gives the user the appropriate information for each employee. Remember that `showOne()` is a method of the `empObject()` constructor and will be called for one specific instance of the custom object.

One way to write this code is as follows:

```
function showOne() {
   var str = "Name: " + this.name + "\n";
   str += "Age: " + this.age + "\n";
   str += "Department: " + this.department;
   alert(str);
}
```

12. Consider the following form:

```
<FORM NAME="myForm">
<INPUT TYPE="text" NAME="name">
<BR>
<INPUT TYPE="button" VALUE="Get Department"
onClick="getDepartment(this.form);">
</FORM>

function getDepartment(form) {

}
```

Using this form, complete the `getDepartment()` function so that users can enter a name in the text box and receive an alert dialog box informing them of the employee's department. Add functionality that informs users if a match is not found for the entered name.

One way to write this code is as follows:

```
function getDepartment(form) {
var name = form.name.value.toLowerCase();
var num = employees.length;
var i = 0;
while(i < num && employees[i].name.toLowerCase() != name){
    i++;
 }
    if(i < num) {
       alert(employees[i].name + " is in " + employees[i]. ➘
       department".");}
    else {
       alert(name + " is not in one of our departments.");
    }
    }
```

13. Suppose you need a function named `getAllEmployees()` that returns all instances of employee objects. Using the `empObject` from a previous question, write this function. Use `document.write()` statements to output the data to an HTML page.

    One way to write this code is as follows:

```
function getAllEmployees() {
var num = employees.length;
  for (var i = 0; i < num; i++) {
    document.write("Employee " + (i + 1) + ":<BR>" + ➘
employees[i].name+ "<BR>");
    document.write("Age: " + employees[i].age + "<BR>");
    document.write("Department : " +  employees[i].department + ➘
    "<P>");
  }
}
```

Course Assessment

The following multiple-choice post-course assessment will evaluate your knowledge of the skills and concepts taught in *JavaScript Fundamentals*.

1. Which of the following code segments properly ensures that JavaScript code will not be displayed in the browser window by older browsers?

   a.
   ```
   <!--
   <SCRIPT LANGUAGE="JavaScript">
   // JavaScript code
     var x = 10;
   </SCRIPT>
   -->
   ```

   b.
   ```
   <SCRIPT LANGUAGE="JavaScript">
   <!--
   // JavaScript code
     var x = 10;
   -->
   </SCRIPT>
   ```

   c.
   ```
   <!--
   <SCRIPT LANGUAGE="JavaScript">
   // JavaScript code
     var x = 10;
   </SCRIPT>
   //-->
   ```

   d.
   <SCRIPT LANGUAGE="JavaScript">
   <!--
   // JavaScript code
   * var x = 10;*
   //-->
   </SCRIPT>

2. Which of the following code segments properly specifies JavaScript for an HTML 4.0-compliant document?

   a. *<SCRIPT TYPE="text/javascript">*
   b. `<SCRIPT TEXT="code/javascript">`
   c. `<SCRIPT TYPE="code/javascript">`
   d. `<SCRIPT LANGUAGE="text/javascript">`

3. Which of the following JavaScript code segments correctly concatenates the variables `x` and `y` into a `document.write()` statement?

   a. `var x = 10, y = 20;`
      `document.write("x is " & x & " and y is " & y & ".");`

   b. *`var x = 10, y = 20;`*
      `document.write("x is " + x + " and y is " + y + ".");`

   c. `var x = 10, y = 20;`
      `document.write(x is & " x " & and y is & " y " & .);`

   d. `var x = 10, y = 20;`
      `document.write("x is" + x + "and y is" + y + ".");`

4. What is the return value when a user selects the Cancel button from a JavaScript confirm dialog box?

   a. true
   b. null
   c. *false*
   d. There is no return value.

5. A JavaScript developer wants to use a confirm dialog box to verify that a user indeed meant to click a particular link. Which of the following code segments properly scripts such an <A> tag?

   a. `<A HREF="http://www.ciwcertified.com/"`
      `onClick="confirm("Proceed?");">Visit CIW</A>`

   b. *`<A HREF="http://www.ciwcertified.com/"`*
      `onClick="return confirm('Proceed?');">Visit CIW</A>`

   c. `<A HREF="http://www.ciwcertified.com/"`
      `onClick="return confirm("Proceed?");">Visit CIW</A>`

   d. `<A HREF="http://www.ciwcertified.com/"`
      `onClick="confirm('Proceed?');">Visit CIW</A>`

6. What is the output of the following JavaScript code segment?

```
var x = 30,  y = 20;

document.write(++x * y--);
```

   a. 589
   b. 570
   c. 651
   d. *620*

7. Which of the following JavaScript code segments properly calls and passes arguments to a function named myFunction()?

   a. `call myFunction(text  more text);`
   b. *`myFunction("text", "more text");`*
   c. `invoke myFunction(text, more text);`
   d. `myFunction("text"; "more text");`

8. Which of the following JavaScript event handlers is invoked when a cursor is placed in a `text` object?

   a. `onSelect`
   b. `onChange`
   c. `onClick`
   d. *`onFocus`*

9. Which of the following code segments properly scripts a JavaScript `for` statement?

   a.
   ```
   for (var i = 0, i <= 10, i--) {
     document.write("loop number " + i + "<BR>");
   }
   ```

   b.
   ```
   for (var i = 0; i >= 10; i++) {
     document.write("loop number " + i + "<BR>");
   }
   ```

   c.
   ```
   for (var i = 0; i <= 10; i++) {
     document.write("loop number " + i + "<BR>");
   }
   ```

   d.
   ```
   for (var i = 0, i >= 0, i--) {
     document.write("loop number " + i + "<BR>");
   }
   ```

10. Which of the following code segments correctly scripts a JavaScript `do...while` statement?

   a.
   ```
   var i = 1;
   do {
     document.write("loop number " + i + "<BR>");
     i--;
   while(i < 10);
   }
   ```

   b.
   ```
   var i = 1;
   do {
     document.write("loop number " + i + "<BR>");
     i++;
   } while(i < 10);
   ```

   c.
   ```
   var i = 1;
   do {
     document.write("loop number " + i + "<BR>");
     i++;
   while(i < 10)
   }
   ```

   d.
   ```
   var i = 1;
   do {
     document.write("loop number " + i + "<BR>");
   } while(i < 10);
   ```

11. Which of the following JavaScript code segments correctly opens a new window?

    a. `var myWindow`
       `myWindow = window.open("newWin.htm";"myWin";"scrollbars=1,menu=1");`

    b. `var myWindow`
       `myWindow = window.open("newWin.htm","my Win", scrollbar=1,menus=1");`

    c. `var myWindow`
       `myWindow = window.open("newWin.htm";"myWin";scrollbars=1;menus=1");`

    d. *`var myWindow`*
       `myWindow = window.open("newWin.htm","myWin",scrollbars=1,menu=1");`

12. Which of the following properties returns the date and time that a `document` object was last revised?

    a. *`lastModified`*
    b. `lastChanged`
    c. `lastModification`
    d. `lastAltered`

13. Which of the following code segments correctly uses JavaScript to preload images for an HTML page?

    a.
    ```
    if (document.images) {

        var pic1, pic2;

        pic1 = new image();
        pic1.source = "images/pic1.gif";

        pic2 = new image();
        pic2.source = "images/pic2.gif";
      }
    ```

    b.
    ```
    if (document.images) {

        var pic1, pic2;

        pic1 = new Image();
        pic1.src = "images/pic1.gif";

        pic2 = new Image();
        pic2.src = "images/pic2.gif";
      }
    ```

    c.
    ```
    if (document.images) {

        var pic1, pic2;

        pic1 = new Image();
        pic1.source = "images/pic1.gif";

        pic2 = new Image();
        pic2.source = "images/pic2.gif";
      }
    ```

    d.
    ```
    if (document.Images) {

        var pic1, pic2;

        pic1 = new image();
        pic1.src = "images/pic1.gif";

        pic2 = new image();
        pic2.src = "images/pic2.gif";
      }
    ```

14. Which of the following JavaScript code segments correctly applies `String` object formatting methods to a string?

    a.
    ```
    var aString = "test text.";
    aString.toUpperCase();
    aString.italics();
    document.write(aString);
    ```

    b.
    var aString = "test text.";
    aString = aString.toUpperCase().italics();
    document.write(aString);

    c.
    ```
    var aString = "test text.";
    aString.toUpperCase(aString);
    aString.italics(aString);
    document.write(aString);
    ```

    d.
    ```
    var aString = "test text.";
    aString.toUpperCase(aString).italic(aString);
    document.write(aString);
    ```

15. Which of the following JavaScript code segments correctly accesses the user's entry in a `text` object named `myTextBox` in a form named `myForm`?

    a.
    ```
    var myVar;
    myVar = document.myForm.myTextbox.text;
    ```

    b.
    ```
    var myVar;
    myVar = document.myForm.myTextbox.input;
    ```

    c.
    var myVar;
    myVar = document.myForm.myTextbox.value;

    d.
    ```
    var myVar;
    myVar = document.myForm.myTextbox.content;
    ```

16. Which of the following JavaScript code segments properly assigns a Boolean value to the `myVar` variable indicating the state of a `checkbox` object? Assume that the `checkbox` object is the second form element defined in the first form of an HTML document.

    a.
    ```
    var myVar;
    myVar = document.forms[1].elements[2].selected;
    ```

    b.
    var myVar;
    myVar = document.forms[0].elements[1].checked;

    c.
    ```
    var myar;
    myVar = document.forms[0].elements[1].selected;
    ```

    d.
    ```
    var myar;
    myVar = document.forms[1].elements[2].checked;
    ```

17. Which of the following JavaScript code segments properly accesses the numeric value returned when a user selects an option from a `select` object named `mySelect` in a form named `myForm`?

a.
```
var myar;
myVar = document.myForm.mySelect.value;
```

b.
```
var myar;
myVar = document.myForm.mySelect.selectedInt;
```

c.
```
var myar;
myVar = document.myForm.mySelect.integer;
```

d. *var myar;*
myVar = document.myForm.mySelect.selectedIndex;

18. Which of the following JavaScript code segments correctly determines which radio button has been selected in a radio button group named `myRadio` in a form named `myForm`?

a.
```
var len = myRadio.length;
var myVar = "";
for(var idx = 0; idx < len; idx++) {
  if(document.myForm.myRadio[idx].selected) {
    myVar = document.myForm.myRadio.value;
  }
}
```

b. *var len = myRadio.length;*
var myVar = "";
for(var idx = 0; idx < len; idx++) {
if(document.myForm.myRadio[idx].checked) {
myVar = document.myForm.myRadio[idx].value;
}
}

c.
```
var len = myRadio.length;
var myVar = "";
for(var idx = 0; idx < len; idx++) {
  if(document.myForm.myRadio[idx].selected) {
    myVar = document.myForm.myRadio[idx].value;
  }
}
```

d.
```
var len = myRadio.length;
var myVar = "";
for(var idx = 0; idx < len; idx++) {
  if(document.myForm.myRadio.checked) {
    myVar = document.myForm.myRadio[idx].value;
  }
}
```

19. Which of the following JavaScript code segments indicates that it is not true that the variable `myVar` is zero or the variable `myVar2` is less than or equal to 20?

a. `!(myVar == 0 || myVar2 =< 20);`
b. `!(myVar != 0) && myVar2 <= 20;`
c. *!(myVar == 0) || myVar2 <= 20;*
d. `myVar != 0 && myVar2 =< 20;`

20. Which of the following code segments correctly creates and populates a JavaScript array?

    a.
    ```
    var myArray = new Array[];
    myArray(0) = "value 1";
    myArray(2) = "value 2";
    ```

    b.
    ```
    var myArray = new Array();
    myArray(0) = "value 1";
    myArray(2) = "value 2";
    ```

    c.
    ```
    var myArray = new Array[];
    myArray[0] = "value 1";
    myArray[2] = "value 2";
    ```

    d.
    ```
    var myArray = new Array();
    myArray[0] = "value 1";
    myArray[2] = "value 2";
    ```

21. What value does `myStr` hold after execution of the following JavaScript code segment?

    ```
    var myStr = "test string".substring(6, 2);
    ```

    a. *st s*
    b. tr
    c. est s
    d. st st

22. What value does `myVar` hold after execution of the following JavaScript code segment?

    ```
    var myVar = "test string".indexOf("st", 3);
    ```

    a. 3
    b. *5*
    c. 6
    d. –1

23. What value does `myVar` hold after execution of the following JavaScript code segment?

    ```
    var myVar = "test string".indexOf("sts");
    ```

    a. -1
    b. 2
    c. 4
    d. 3

24. In general terms, what is the value of `myVar` after execution of the following JavaScript code segment?

    ```
    var myVar = navigator.appVersion.substring(0, 1);
    ```

    a. The first letter of the name of the browser.
    b. The second letter of the name of the browser.
    c. The version number of the browser.
    d. *The first number in the version number of the browser.*

Appendixes

** Appendix found on Supplemental CD-ROM*

Glossary

argument—A value or expression containing data or code that is passed on to a function or procedure.

calling statement—A statement that transfers program execution to a subroutine, procedure or function. When the subroutine is complete, execution transfers back to the command following the call statement.

concatenation—Synthesis of code to simplify it and reduce duplication; linking two or more units of information, such as strings or files, to form one unit.

constructor—A special function that enables you to create instances of custom objects.

cookie—Information sent between a server and a client to help maintain state and track user activities. Cookies can reside in memory or be placed on a hard drive in the form of a text file.

digital certificate—An electronic mechanism used to establish the identity of an individual or an organization.

Document Object Model (DOM)—A World Wide Web Consortium specification intended to render an HTML page or XML document as a programmable object.

dot notation—A reference used to access a property or method of an object.

event handler—A JavaScript mechanism for intercepting an event and associating executable code with that event.

floating-point calculation—A calculation in which the decimal point can move as needed to account for significant digits.

function—A named block of code that can be called when needed. In JavaScript, a function can return a value.

graphical user interface (GUI)—A program that provides graphical navigation with menus and screen icons.

method—An action that can be performed by an object.

Multipurpose Internet Mail Extensions (MIME)—E-mail protocol extensions that provide methods for encoding binary data for electronic transmission.

nested statement—A common programming practice of embedding a script block within another script block.

object—A programming function that models the characteristics of abstract or real "objects" using classes.

operand—Data that is to be operated upon or manipulated in some manner.

pass by reference—Mode that describes when values are passed to a function, and the function's parameters receive its argument's actual value.

pass by value—Mode that describes when values are passed to a function, and the function's parameters receive a copy of its argument's value.

Perl—A script programming language commonly used for Web server tasks and CGI programs.

property—A descriptive characteristic of an object, such as color, width or height, that the programmer stipulates in the creation of the object.

REXX—A procedural programming language used to create programs and algorithms.

statement—A single line of code to be executed in a script or program.

Tool Command Language (Tcl)—An interpreted script language used to develop applications such as GUIs, prototypes and CGI scripts.

user agent—The W3C term for any application, such as a Web browser or help engine, that renders HTML for display to users.

value—The specific quality such as color, width or height that belongs to the property of an object.

variable—A named space of memory that holds a value. The value can change depending on a condition, information passed to the program, or by reassignment of a new value.

Index

Supplemental CD-ROM Contents

The *JavaScript Fundamentals* supplemental CD-ROM contains the following files needed to complete the course labs:

JavaScript_Fund_Instructor_CD

Answers	Classroom	Lab Files
Appendix	Handouts	

Answers

ANSWERS_Activity.pdf	ANSWERS_OptionalLab.pdf	ANSWERS_Quiz.pdf
ANSWERS_CourseAssessment.pdf	ANSWERS_PreAssessment.pdf	ANSWERS_Review.pdf
ANSWERS_Lab.pdf		

Appendix

Appendix_A.pdf	Appendix_F.pdf	Appendix_K.pdf
Appendix_B.pdf	Appendix_G.pdf	Appendix_L.pdf
Appendix_C.pdf	Appendix_H.pdf	Appendix_M.pdf
Appendix_D.pdf	Appendix_I.pdf	Appendix_N.pdf
Appendix_E.pdf	Appendix_J.pdf	

Classroom

Syllabus	JavaScript_Fund_ClassroomSetup.pdf
Implementation Table	JavaScript_Fund_Slideshow.pdf

Classroom\Syllabus

CIW_Web_Languages_Syllabus_Academic_10week.doc
CIW_Web_Languages_Syllabus_Academic_16week.doc

Classroom\Implementation Table

JavaScript_Fnd_ImplementationTable.pdf

Handouts

HANDOUTS_Activity.pdf	HANDOUTS_OptionalLab.pdf	HANDOUTS_Quiz.pdf
HANDOUTS_CourseAssessment.pdf		

Lab Files

Completed_Labs	Student_Files

Lab Files\Completed_Labs

Appendix	Lesson 3	Lesson 7
Lesson 1	Lesson 4	Lesson 8
Lesson 10	Lesson 5	Lesson 9
Lesson 2	Lesson 6	

Lab Files\Completed_Labs\Appendix

Frames review A	errorcity.htm	sort.htm
Frames review B	imageMap.htm	tryCatch.htm
images	passValRef.htm	

Lab Files\Completed_Labs\Lesson 1

lab1-1.htm

Lab Files\Completed_Labs\Lesson 10

images	CIWProducts2.htm	optionalLab10-1.htm
CIWProducts.htm	optionalLab10-1.1.htm	

Lab Files\Completed_Labs\Lesson 2

lab2-1.htm	lab2-4.htm	lab2-6.htm
lab2-2.htm	lab2-5.htm	optionalLab2-1.htm
lab2-3.htm		

Lab Files\Completed_Labs\Lesson 3

lab3-1.htm	lab3-2.htm	optionalLab3-1.htm
lab3-2.1.htm	lab3-3.htm	optionalLab3-2.htm
lab3-2.2.htm	optionalLab3-1.1.htm	

Lab Files\Completed_Labs\Lesson 4

lab4-1.htm	lab4-3.htm	optionalLab4-1.1.htm
lab4-2.1.htm	lab4-4.htm	optionalLab4-1.htm
lab4-2.htm	lab4-5.htm	

Lab Files\Completed_Labs\Lesson 5

Images	Lab5-3.htm	ohwell.htm
Lesson 5 Application Project	Lab5-4.1.htm	optionalLab5-1.htm
congrats.htm	Lab5-4.htm	optionalLab5-2.htm
lab5-1.htm	Lab5-5.htm	JavaScrpt_Fnd_v5_07_Lab_5_2_2KB
Lab5-2.htm		

Lab Files\Completed_Labs\Lesson 5\Lesson 5 Application Project

images	lesson5AppProj.htm

Lab Files\Completed_Labs\Lesson 6

Images	lab6-2.1.htm	lab6-4.htm
Lesson_6_Application_Project	lab6-2.htm	lab6-5.htm
lab6-1.htm	lab6-3.htm	optionalLab6-1.htm

Lab Files\Completed_Labs\Lesson 6\Lesson_6_Application_Project

lesson6AppProj.htm

Lab Files\Completed_Labs\Lesson 7

Lesson_7_Application_Project	lab7-2.1.htm	Lab7-3.htm
lab7-1.1.htm	Lab7-2.htm	Lab7-4.htm
Lab7-1.2.htm	Lab7-3.1.htm	optionalLab7-1.htm
Lab7-1.htm		

Lab Files\Completed_Labs\Lesson 7\Lesson_7_Application_Project

lesson7AppProj.htm

Lab Files\Completed_Labs\Lesson 8

lab8-1.htm	optionalLab8-1.1.htm	optionalLab8-1.htm
lab8-2.htm		

Lab Files\Completed_Labs\Lesson 9

Lab_9-1	Lesson_9_Application_Project	lab9-3.htm
Lab_9-2	OptionalLab_9-1	

Lab Files\Completed_Labs\Lesson 9\Lab_9-1

images	default.htm	new1.htm
banner.htm	major.htm	new1.htm
banner2.htm	minor.htm	

Lab Files\Completed_Labs\Lesson 9\Lab_9-2

images	default.htm	new1.htm
banner.htm	major.htm	toc.htm
banner2.htm	minor.htm	

Lab Files\Completed_Labs\Lesson 9\Lesson_9_Application_Project

classForm.htm	classList.htm

Lab Files\Completed_Labs\Lesson 9\OptionalLab_9-1

images	main.htm	top.htm

Lab Files\Student_Files

Lesson 1	Lesson 4	Lesson 7
Lesson 10	Lesson 5	Lesson 8
Lesson 2	Lesson 6	Lesson 9
Lesson 3		

Lab Files\Student_Files\Lesson 1

lab1-1.htm

Lab Files\Student_Files\Lesson 10

images	CIWProducts.htm	optionalLab10-1.htm

Lab Files\Student_Files\Lesson 2

📄 lab2-1.htm	📄 lab2-4.htm	📄 lab2-6.htm
📄 lab2-2.htm	📄 lab2-5.htm	📄 optionalLab2-1.htm
📄 lab2-3.htm		

Lab Files\Student_Files\Lesson 3

📄 lab3-1.htm	📄 lab3-2.htm	📄 optionalLab3-1.htm
📄 lab3-2.1.htm	📄 lab3-3.htm	📄 optionalLab3-2.htm
📄 lab3-2.2.htm		

Lab Files\Student_Files\Lesson 4

📄 lab4-1.htm	📄 lab4-3.htm	📄 optionalLab4-1.1.htm
📄 lab4-2.1.htm	📄 lab4-4.htm	📄 optionalLab4-1.htm
📄 lab4-2.htm	📄 lab4-5.htm	

Lab Files\Student_Files\Lesson 5

📁 images	📄 Lab5-3.htm	📄 ohwell.htm
📄 congrats.htm	📄 Lab5-4.htm	📄 optionalLab5-1.htm
📄 Lab5-1.htm	📄 Lab5-5.htm	📄 optionalLab5-2.htm
📄 Lab5-2.htm	📄 lesson5AppProj.htm	

Lab Files\Student_Files\Lesson 6

📁 Images	📄 lab6-2.1.htm	📄 lab6-4.htm
📁 Lesson_6_Application_Project	📄 lab6-2.htm	📄 lab6-5.htm
📄 lab6-1.htm	📄 lab6-3.htm	📄 optionalLab6-1.htm

Lab Files\Student_Files\Lesson 6\Lesson_6_Application_Project

📄 lesson6AppProj.htm

Lab Files\Student_Files\Lesson 7

📁 Lesson_7_Application_Project	📄 Lab7-2.htm	📄 Lab7-3.htm
📄 Lab7-1.2.htm	📄 Lab7-3.1.htm	📄 Lab7-4.htm
📄 Lab7-1.htm		📄 optionalLab7-1.htm

Lab Files\Student_Files\Lesson 7\Lesson_7_Application_Project

📄 lesson7AppProj.htm

Lab Files\Student_Files\Lesson 8

📄 lab8-1.htm	📄 optionalLab8-1.1.htm	📄 optionalLab8-1.htm
📄 lab8-2.htm		

Lab Files\Student_Files\Lesson 9

📁 Lab_9-1	📁 Lesson_9_Application_Project	📄 lab9-3.htm
📁 Lab_9-2	📁 OptionalLab_9-1	

🗁Lab Files\Student_Files\Lesson 9\Lab_9-1

🗀 images	🗎 default.htm	🗎 new1.htm
🗎 banner.htm	🗎 major.htm	🗎 new1.htm
🗎 banner2.htm	🗎 minor.htm	

🗁Lab Files\Student_Files\Lesson 9\Lab_9-2

🗀 images	🗎 default.htm	🗎 new1.htm
🗎 banner.htm	🗎 major.htm	🗎 toc.htm
🗎 banner2.htm	🗎 minor.htm	

🗁Lab Files\Student_Files\Lesson 9\Lesson_9_Application_Project

🗎 classForm.htm	🗎 classList.htm

🗁Lab Files\Student_Files\Lesson 9\OptionalLab_9-1

🗀 images	🗎 main.htm	🗎 top.htm

Handouts: Optional Labs

Optional Lab 1-1: Experimenting with JavaScript

This lab will provide an opportunity to experiment with the JavaScript-enabled file that you created in Lab 1-1.

1. **Editor:** Open the **lab1-1.htm** file. Save the file as **optionalLab1-1.htm**. Experiment with the `document.write()` statements by adding literal text to them. The `document.write()` statement can receive text or any HTML that you may want to use. At this point, there is no right or wrong action to take. Experiment with the document. It will help you become familiar with adding or deleting code inside a JavaScript block.
2. **Editor:** After each change to the optionalLab1-1.htm file, save the file.
3. **Browser:** Browse to **optionalLab1-1.htm**. View your changes. If an error occurs, return to the file in your editor and see whether you can determine why the error occurred.

This lab provided you with an opportunity to experiment with a `document.write()` statement. You added text and HTML to the statements and viewed the output using the browser of your choice.

Optional Lab 2-1: Using the JavaScript *onUnload* event handler and inline scripting

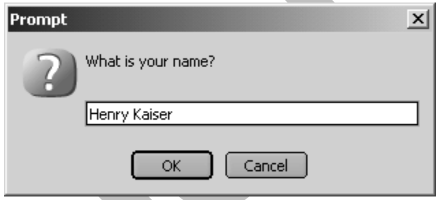
In this lab, you will obtain the user's name as he or she enters the page, greet the user by name, write that name on the page to personalize it, then use the name one last time as the page is unloaded. You will use both inline scripting and the `onUnload` event handler.

1. **Editor:** Open **optionalLab2-1.htm** from the Lesson 2 folder of the Student_Files directory.

2. **Editor:** Enter source code in the `<BODY>` tag. Add an **`onUnload`** event handler that calls an `alert()` box. The text should read `Goodbye`, with the `userName` variable concatenated into the string, then the text `Hurry back!`

3. **Editor:** Save **optionalLab2-1.htm**.

4. **Browser:** Open **optionalLab2-1.htm**. You should see a series of dialog and alert boxes. Enter your name in the prompt dialog box and click **OK** as shown in Figure OL2-1.

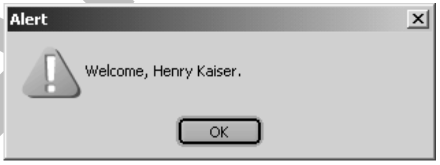


What is your name?

Henry Kaiser

Figure OL2-1: Prompt dialog box

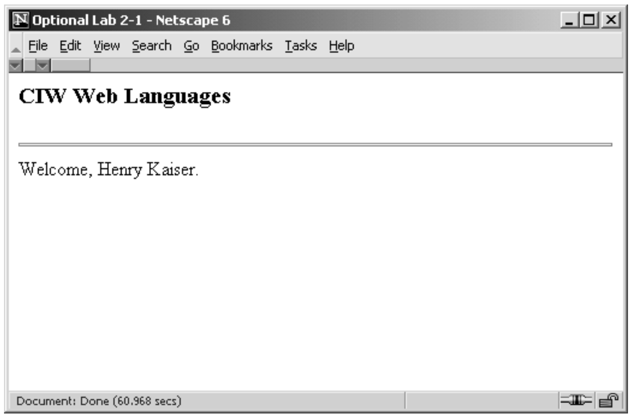
5. **Browser:** The greeting alert box will appear, as shown in Figure OL2-2. Click **OK**.



Welcome, Henry Kaiser.

Figure OL2-2: Alert box

6. **Browser:** The optionalLab2-1.htm page will appear with a personalized greeting, as shown in Figure OL2-3.


Optional Lab 2-1 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help

CIW Web Languages

Welcome, Henry Kaiser.

Document: Done (60.968 secs)

Figure OL2-3: OptionalLab.htm with welcome message

7. **Browser:** Select the **Back** button from your browser's tool bar to navigate to a different page. You should see the alert box shown in Figure OL2-4.

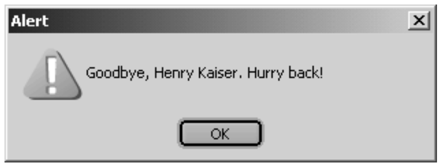

Alert
Goodbye, Henry Kaiser. Hurry back!

Figure OL2-4: Goodbye alert box

8. **Browser:** After clicking **OK** on the alert dialog box, your screen will show the page to which you navigated.

This optional lab provided an opportunity to add inline scripting using the `onUnload` event handler. You will learn more about inline scripting, events and event handlers in the coming lessons.

Optional Lab 3-1: Using a JavaScript conversion function

This optional lab will provide an opportunity to use the JavaScript `parseFloat()` conversion function.

1. **Editor:** Open **optionalLab3-1.htm** from the Lesson 3 folder of the Student_Files directory.

2. **Editor:** In the first <SCRIPT> block, examine the `addNumbers()` function that has been defined for you.

```
function addNumbers (arg1, arg2) {
  var result = arg1 + arg2;
  return result;
}
```

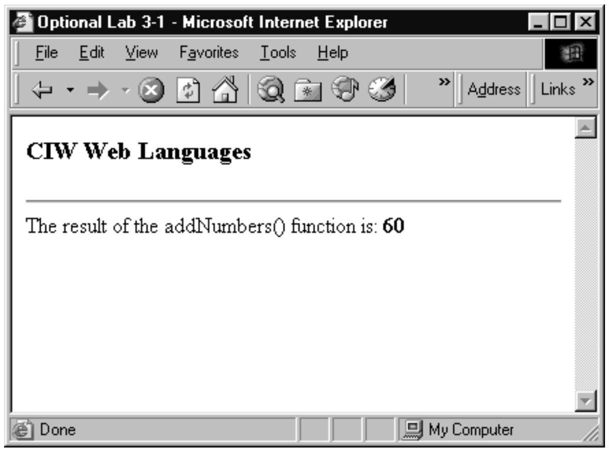
3. **Editor:** This simple function receives two values, adds them together and returns the result. In the second <SCRIPT> block, examine the following code:

```
alert("Please enter a numerical value in each of the ➘
following prompt dialog boxes. \nThe two numbers will ➘
be added together.");

var num1 = prompt("Please enter a numerical value.","");
var num2 = prompt("Please enter a numerical value.","");

document.write("The result of the addNumbers() ➘
function is: <B>" + addNumbers(num1, num2) + "</B><P>");
```

4. **Editor:** Recall that the return value of a `prompt()` method is a string. Determine the various places in the code to use the `parseFloat()` function to convert the user's input to a numerical value. The `parseFloat()` function can be used in several places to achieve the desired result.

5. **Editor:** Save **optionalLab3-1.htm**.

6. **Browser:** Open **optionalLab3-1.htm**. You will see an alert dialog box informing the user of the purpose of the page. You will then see two prompt dialog boxes asking for user input. After you supply the input, your screen should resemble Figure OL3-1, depending on the data input.


Optional Lab 3-1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
Done
My Computer

CIW Web Languages

The result of the addNumbers() function is: **60**

Figure OL3-1: OptionalLab3-1.htm

7. **Browser:** Test the page to ensure that the `addNumbers()` function returns the proper result.

This lab provided an opportunity to use a JavaScript conversion function. You saw that the `parseFloat()` function is used to convert string values to numerical values.

Optional Lab 3-2: Using JavaScript event handlers

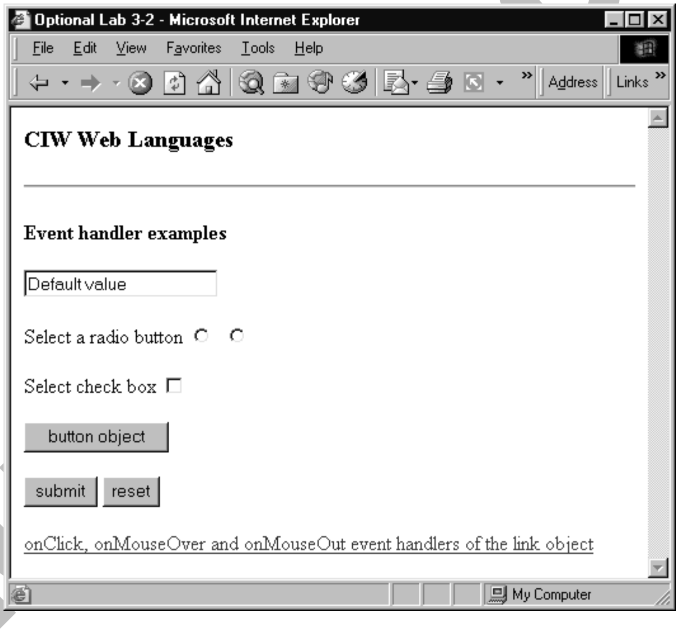
This optional lab will provide an opportunity to use JavaScript event handlers.

1. **Editor:** Open **optionalLab3-2.htm** from the Lesson 3 folder of the Student_Files directory.

2. **Editor:** Locate the HTML comments placed before each HTML object such as the following comment: **`<!-- Add onSubmit and onReset event handlers of the form object -->`** Add the appropriate event handler inline with the appropriate HTML tags. In cases in which more that one event handler is mentioned, add the event handlers in the order in which they are listed in the HTML comment.

3. **Editor:** Save **optionalLab3-2.htm**.

4. **Browser:** Open **optionalLab3-2.htm**. Your screen should resemble Figure OL3-2.


Optional Lab 3-2 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
My Computer
CIW Web Languages

Event handler examples

Default value

Select a radio button

Select check box

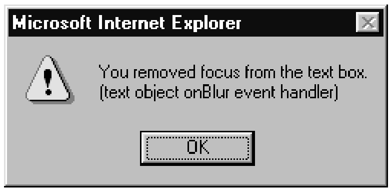
button object

submit reset

onClick, onMouseOver and onMouseOut event handlers of the link object

Figure OL3-2: OptionalLab3-2.htm

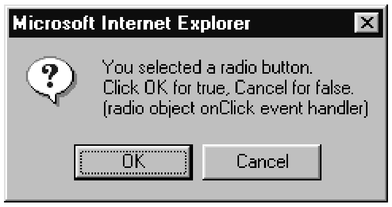
5. **Browser:** Click inside the text box. Removing the cursor from the text box should result in an alert dialog box as shown in Figure OL3-3.



You removed focus from the text box.
(text object onBlur event handler)

Figure OL3-3: Alert dialog box

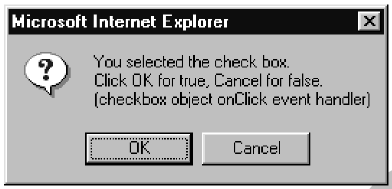
6. **Browser:** Clicking a radio button should result in a confirm dialog box as shown in Figure OL3-4.


Microsoft Internet Explorer

You selected a radio button.
Click OK for true, Cancel for false.
(radio object onClick event handler)

Figure OL3-4: Confirm dialog box

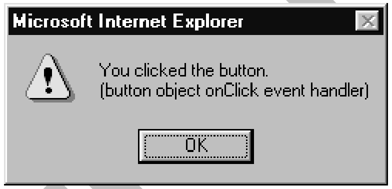
7. **Browser:** Clicking the check box should result in a confirm dialog box as shown in Figure OL3-5.


Microsoft Internet Explorer

You selected the check box.
Click OK for true, Cancel for false.
(checkbox object onClick event handler)

Figure OL3-5: Confirm dialog box

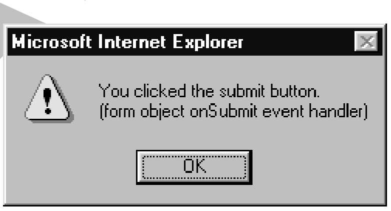
8. **Browser:** Clicking the **button object** button should result in an alert dialog box as shown in Figure OL3-6.


Microsoft Internet Explorer

You clicked the button.
(button object onClick event handler)

Figure OL3-6: Alert dialog box

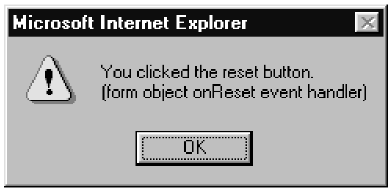
9. **Browser:** Clicking the **submit** button should result in an alert dialog box as shown in Figure OL3-7.


Microsoft Internet Explorer

You clicked the submit button.
(form object onSubmit event handler)

Figure OL3-7: Alert dialog box

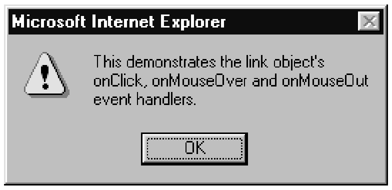
10. **Browser:** Clicking the **reset** button should result in an alert dialog box as shown in Figure OL3-8.


Microsoft Internet Explorer

You clicked the reset button.
(form object onReset event handler)

Figure OL3-8: Alert dialog box

11. **Browser:** Clicking the link should result in an alert dialog box as shown in Figure OL3-9.

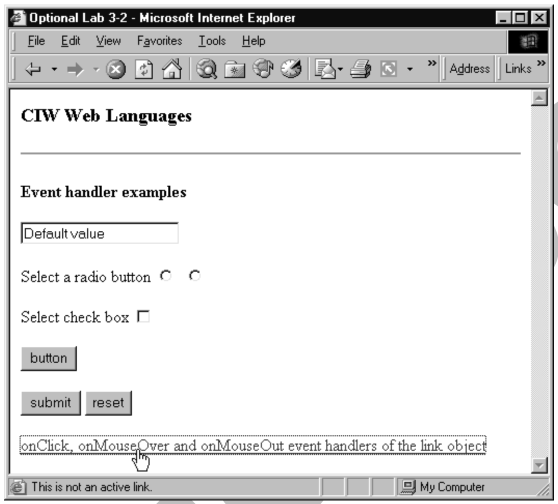


Microsoft Internet Explorer

This demonstrates the link object's onClick, onMouseOver and onMouseOut event handlers.

Figure OL3-9: Alert dialog box

12. **Browser:** Placing the cursor over the link should result in a message in the browser's status bar as shown in Figure OL3-10.



Optional Lab 3-2 - Microsoft Internet Explorer

CIW Web Languages

Event handler examples

Default value

Select a radio button

Select check box

onClick, onMouseOver and onMouseOut event handlers of the link object

This is not an active link.

Figure OL3-10: Message in status bar

13. **Browser:** Removing the cursor from the link should cause the message displayed in the status bar of Figure OL3-10 to disappear.

14. **Browser:** Experiment with the various objects on the page. In particular, note the behavior of the radio buttons, check boxes and confirm dialog boxes that are launched when these objects are selected.

This lab provided an opportunity to use and observe the behavior of commonly used JavaScript event handlers. Several concepts were introduced in this lab that have yet to be discussed in this course. In particular, you saw the `status` property of the `window` object manipulated with inline script. This syntax will be discussed in detail in another lesson.

Optional Lab 4-1: Using a *switch* statement

This optional lab will provide an opportunity to use a `switch` statement.

1. **Editor:** Open **optionalLab4-1.htm** from the Lesson 4 folder of the Student_Files directory.
2. **Editor:** A function named `switchTest()` has been started for you. A variable named `choice` is defined and receives its value from a drop-down menu selection.
3. **Editor:** Create a `switch` statement. The test expression is the `choice` variable. The text after the `case` keywords is the text from the last three drop-down menu items, **Choice 1**, **Choice 2** and **Choice 3**. Create an alert dialog box that reflects back to the user the item selected, as code to execute after each `case` statement.
4. **Editor:** Add a `default` clause that outputs an alert dialog box asking the user to make a selection from the drop-down menu.
5. **Editor:** Save **optionalLab4-1.htm**.
6. **Browser:** Open **optionalLab4-1.htm**. Your screen should resemble Figure OL4-1.

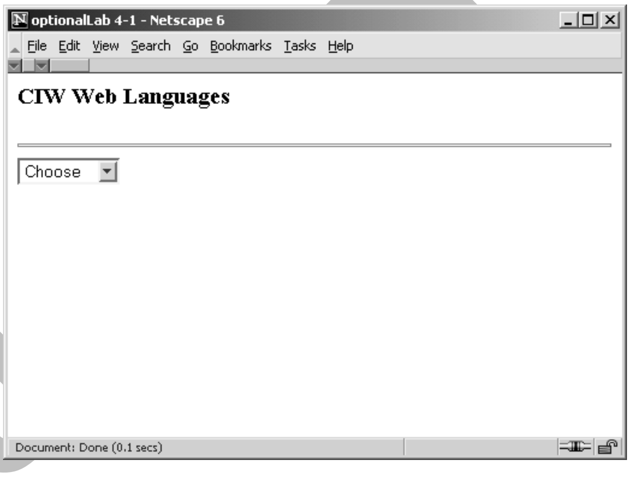

optionalLab 4-1 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
CIW Web Languages
Choose
Document: Done (0.1 secs)

Figure OL4-1: OptionalLab4-1.htm

7. **Browser:** Select an item from the drop-down menu. You should see an alert dialog box similar to Figure OL4-2, depending on the selection.

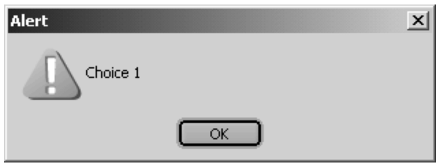
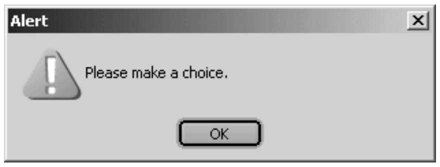


Figure OL4-2: Alert dialog box

8. **Browser:** Click **OK** to close the alert dialog box. Select the first drop-down menu item. This action should trigger the `default` clause in your code. An alert similar to Figure OL4-3 should appear.



Please make a choice.

Figure OL4-3: Alert dialog box

9. **Browser:** Test the page to ensure that the proper alert dialog box appears for all situations.

10. **Editor:** Open **optionalLab4-1.1.htm** from the Lesson 4 folder of the Student_Files directory. This file will provide an opportunity to create a `switch` statement that executes the same code for two different `case` expressions.

11. **Editor:** A function named `switchTest()` has been started for you. A variable named `choice` is defined and receives its value from a drop-down menu selection. As in optionalLab4-1.htm, create a `switch` statement. The test expression is the `choice` variable. The text after the `case` keywords is the text from the last four drop-down menu items, **Los Angeles**, **San Francisco**, **Chicago** and **New York City**.

12. **Editor:** As the code to execute for each `case` statement, create an alert dialog box that displays for the user the state that each city is in. These states are **California** for both Los Angeles and San Francisco, **Illinois** for Chicago and **New York** for New York City. Because the same code will execute for Los Angeles and San Francisco, create a fall-through effect for those two `case` statements.

13. **Editor:** As in optionalLab4-1.htm, create a `default` clause that outputs an alert dialog box that asks the user to make a choice.

14. **Editor:** Save **optionalLab4-1.1.htm**.

15. **Browser:** Open **optionalLab4-1.1.htm**. Your screen should resemble Figure OL4-4.

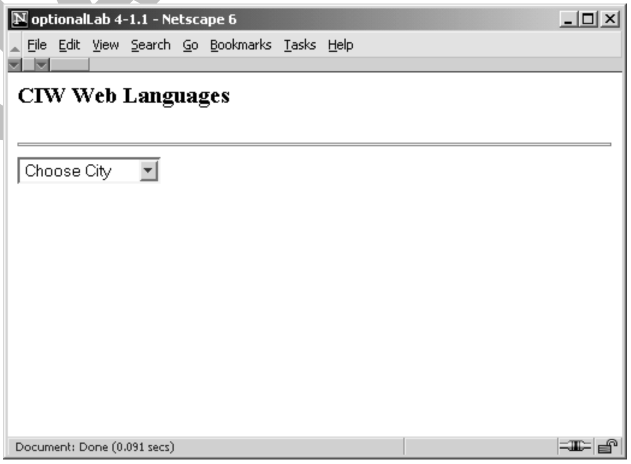
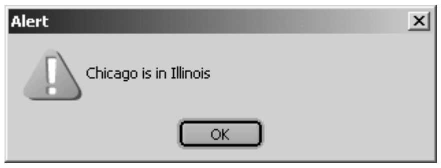

optionalLab 4-1.1 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
Choose City
Document: Done (0.091 secs)
CIW Web Languages

Figure OL4-4: OptionalLab4-1.1.htm

16. **Browser:** Select **Chicago** from the drop-down menu. An alert dialog box similar to Figure OL4-5 should appear.



Chicago is in Illinois

Figure OL4-5: Alert dialog box

17. **Browser:** Test the page to ensure that the proper alert dialog box appears for each situation.

In this lab, you used the `switch` statement. You learned that the `switch` construct is an effective way to create code that executes based on the comparison of one expression against many expressions. You also learned how to create a fall-through effect in a `switch` statement, so that the same code will execute for different values.

Optional Lab 5-1: Using the *location* object with a condition

In this optional lab, you will work with a script you used previously. Using the quiz you examined in an earlier lab, you will use the `location` object to send the user to one of two pages, depending upon the number of correct answers given.

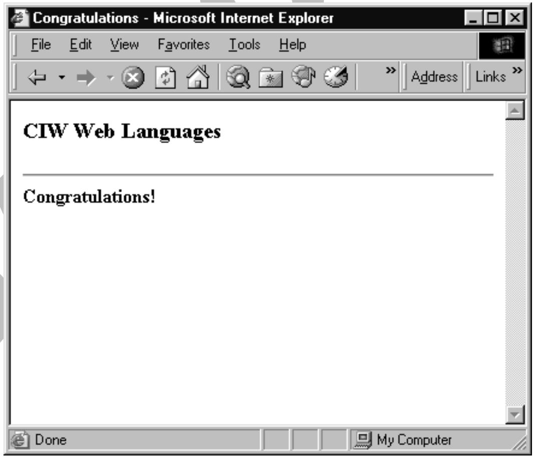
1. **Editor:** Open **optionalLab5-1.htm** from the Lesson 5 folder of the Student_Files directory.

2. **Editor:** Add a `location.href` statement and a conditional assignment operator to the code, so that three correct answers will send the user to a "congratulations" page. If users answer fewer than three questions correctly, send them to a "better luck next time" page. These HTML pages have been provided for you and are located in the Lesson 5 folder of the Student_Files directory. The file names are **congrats.htm** and **ohwell.htm**. These files will provide the values for the `location.href` statements.

3. **Editor:** If you are unsure where to place the `location.href` code, consider the point where it would be logical to send a user to another page. For example, would it be after the first prompt? After the first alert? Or perhaps after the third alert?

4. **Editor:** Save **optionalLab5-1.htm**.

5. **Browser:** Open **optionalLab5-1.htm**. Respond correctly to the quiz questions. You should be taken to the page shown in Figure OL5-1.

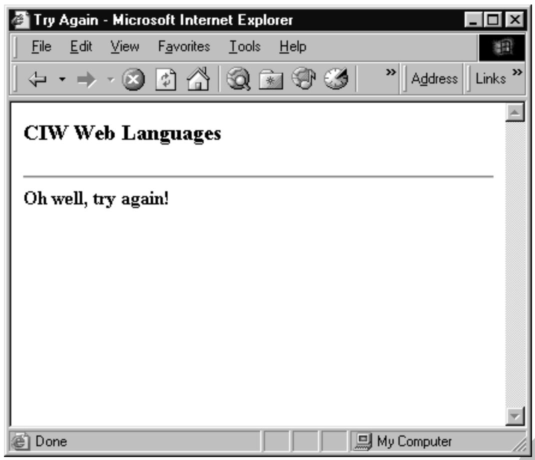

Congratulations - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
Done
My Computer

CIW Web Languages

Congratulations!

Figure OL5-1: Congrats.htm

6. **Browser:** Return to **optionalLab5-1.htm**. Answer at least one of the questions incorrectly. You should be taken to the page shown in Figure OL5-2.


Try Again - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
Done
My Computer

CIW Web Languages

Oh well, try again!

Figure OL5-2: Ohwell.htm

This lab demonstrated the `location` object used for navigational purposes. Remember that `location` object properties also provide access to each portion of a URL. This feature can be useful for applications that need to extract information from URLs.

Optional Lab 5-2: Using the *navigator* object

The following optional lab will demonstrate the type of information you can detect from several `navigator` object properties.

1. **Editor:** Open **optionalLab5-2.htm** from the Lesson 5 folder of the Student_Files directory.

2. **Editor:** Locate the <SCRIPT> block in the <HEAD> section of the file. You will add code to the `showInfo()` function. Concatenate the **`appCodeName`**, **`appName`**, **`appVersion`** and **`userAgent`** properties into the `info` variable.

3. **Editor:** The following code shows the `showInfo()` function before your changes:

```
<SCRIPT LANGUAGE="JavaScript">
<!--

function showInfo() {
  var info="";

  info += "Welcome, " +               ; //Add appCodeName here
  info += " user!\nYou are using the ";
  info +=      + " browser,\nversion "; //Add appName here
  info +=      + ".\nYour user agent "; //Add appVersion here
  info += "information is " +         ; //Add userAgent here

  alert(info);
}

//-->
</SCRIPT>
```

4. **Editor:** Save **optionalLab5-2.htm**.

5. **Browser:** Open **optionalLab5-2.htm** in the Netscape browser. Your screen should resemble Figure OL5-3.

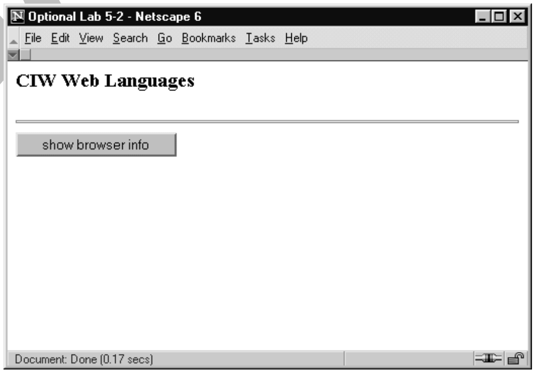
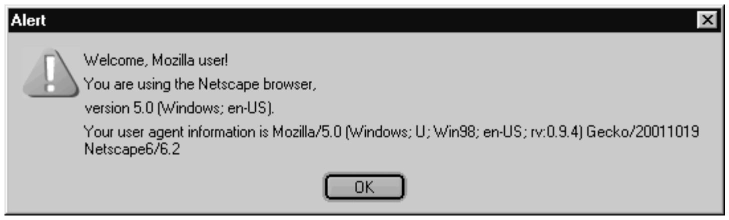

Optional Lab 5-2 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
CIW Web Languages
show browser info
Document: Done (0.17 secs)

Figure OL5-3: OptionalLab5-2.htm

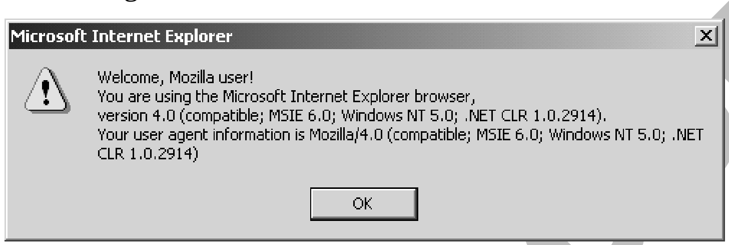
6. **Browser:** Click the **show browser info** button. You should see an alert similar to Figure OL5-4.



```
Alert
Welcome, Mozilla user!
You are using the Netscape browser,
version 5.0 (Windows; en-US).
Your user agent information is Mozilla/5.0 (Windows; U; Win98; en-US; rv:0.9.4) Gecko/20011019
Netscape6/6.2
```

Figure OL5-4: Alert in Netscape Navigator 6.2

7. **Browser:** Open **optionalLab5-2.htm** in Microsoft Internet Explorer. You should see an alert similar to Figure OL5-5.



```
Microsoft Internet Explorer
Welcome, Mozilla user!
You are using the Microsoft Internet Explorer browser,
version 4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.0.2914).
Your user agent information is Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET
CLR 1.0.2914)
```

Figure OL5-5: Alert in Microsoft Internet Explorer 6.0

This lab demonstrated information that is available when using the `navigator` object. One of the most challenging aspects of Internet application development is creating code that will function in all browsers. When coding for different browsers is a mission-critical concern, the information that the `navigator` object contains is an essential element. As your application development experience grows, you will find many uses for the `navigator` object.

Optional Lab 6-1: Using the *Math* object to generate a random quotation

In this lab, you will use the Math object and two of its methods. The HTML page used for this lab will display random quotations, using the Math object to determine which quotation will appear.

1. **Editor:** Open the **optionalLab6-1.htm** file from the Lesson 6 folder of the Student_Files directory.

2. **Editor:** You will add two lines of code to the randquotes.htm file. Locate the existing <SCRIPT> block in the <BODY> section of the document. Locate the comment that reads: **// Use Math object**.

3. **Editor:** Create a variable named **num**. Assign as its value a random integer between 0 and 9. Use the Math object's **round()** and **random()** methods in tandem to accomplish this task. Remember that the random() method generates a random number between 0 and 1. Because the quotes array contains nine quotations, the random number generated must be multiplied by 9. Use the **round()** method to round that number to the nearest integer.

Hint: The Math.random() expression can be passed as an argument to the Math.round() method.

4. **Editor:** Create a **document.write()** statement that will output an element of the quotes array. Use the **num** variable as the subscript for the quotes array. In other words, the num variable should be placed inside the square brackets after the quotes array name. The following shows the code before your changes:

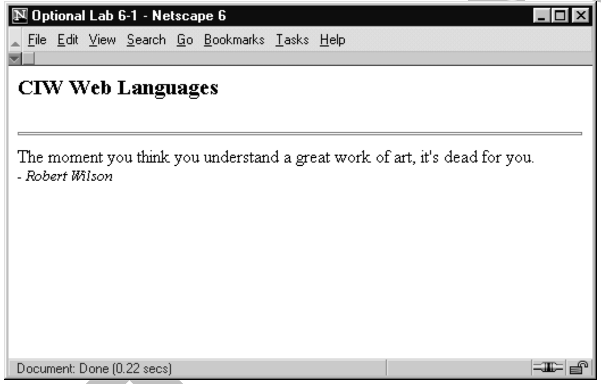
```
<SCRIPT LANGUAGE="JavaScript">
<!--

var quotes = new Array();

quotes[0] = "Every time history repeats itself the price goes➘
up.<BR><small><i>- Anonymous</i></small>";

quotes[1] = "The moment you think you understand a great work➘
of art, it's dead for you.<BR><small><i>- Robert Wilson</small></i>";

quotes[2] = "To love one person with a private love is poor ➘
and miserable; to love all is glorious.<BR><small><i>- Thomas➘
Traherne</small></i>";

quotes[3] = "Every violation of truth is not only a sort of ➘
suicide in the liar, but is a stab at the health of human➘
society.<BR><small><i>- Ralph Waldo Emerson</i></small>";

quotes[4] = "Man is to be found in reason, God in the ➘
passions.<BR><small><i>- G. C. Lichtenberg</i></small>";

quotes[5] ="Great innovations should not be forced on slender➘
majorities.<BR><small><i>- Thomas Jefferson</i></small>";

quotes[6] = "In this world nothing can be said to be certain,➘
except death and taxes.<BR><small><i>- Benjamin ➘
Franklin</i></small>";

quotes[7] = "Nine-tenths of wisdom consists in being wise in ➘
time.<BR><small><i>- Theodore Roosevelt</i></small>";
```

```
quotes[8] = "We have no more right to consume happiness ➘
without producing it than to consume wealth without producing➘
it.<BR><small><i>- George Bernard Shaw</i></small>";

quotes[9] = "So little done, so much to do.<BR><small><i>- ➘
Cecil Rhodes</i></small>";


// Use Math object


// -->
</SCRIPT>
```

5. **Editor:** Save **optionalLab6-1.htm**.

6. **Browser:** Open **optionalLab6-1.htm**. Your screen should resemble

Figure OL6-1. If it does not, verify that the source code you entered is correct.


Optional Lab 6-1 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
Document: Done (0.22 secs)

CIW Web Languages

The moment you think you understand a great work of art, it's dead for you.
- *Robert Wilson*

Figure OL6-1: OptionalLab6-1.htm

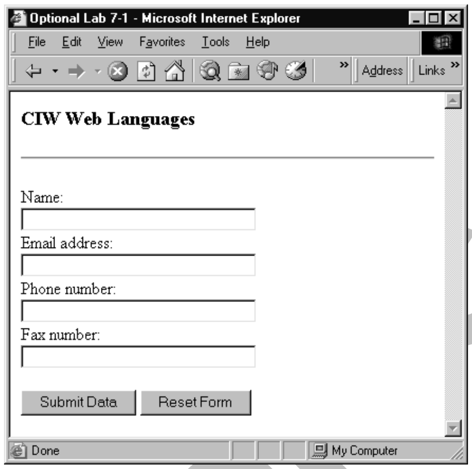
7. **Browser:** Reload the file several times. You will see different quotations. Because the generator is truly random, you may see the same quotation more than once, even subsequently. If you reload enough times, you will eventually see all the quotations.

In this lab, you used the `Math` object's `random()` method to generate a random number between zero and one. You multiplied that random number by nine, then used the `round()` method to create an integer used as the subscript for the `quotes` array. You then used a `document.write()` statement to output a random element from the array.

Optional Lab 7-1: Conducting form validation

In this lab, you will use JavaScript to validate user input to an HTML form.

1. **Editor:** Open the **optionalLab7-1.htm** file from the Lesson_7 folder of the Student_Files directory. The HTML code is already included to create a form that resembles Figure OL7-1.


Optional Lab 7-1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
Submit Data
Reset Form
Done
My Computer

CIW Web Languages

Name:

Email address:

Phone number:

Fax number:

Figure OL7-1: Interactive form

2. **Editor:** A `sumbit` object is scripted to call a function named `checkForm()`. Examine the function invocation statement:

```
<INPUT TYPE="submit" VALUE="Submit Data"
onClick="return checkForm(this.form);">
```

3. **Editor:** Note the use of the `return` keyword. The `checkForm()` function will be written to return either a true or a false value. If a `true` value is returned, the form will be submitted. A `false` return value will cancel the form submission. The argument `this.form` passes the form's name/value pairs to the function

4. **Editor:** In the `checkForm()` function, a variable is created for you. The `len` variable is assigned the return value of the form object's `length` property. This variable will be used in a `for` loop to determine how many times the loop will execute.

5. **Editor:** Locate the comment that reads as follows: `// Create for loop`. Create a `for` loop. Use a loop counter variable that starts at zero. Use the `len` variable in the loop's logical expression. Inside the `for` loop, create an `if` statement. As the condition for the `if` statement, use the `elements` property of the `form` object to determine if each form element is either `null` or empty. Use the logical OR operator (||) for this task.

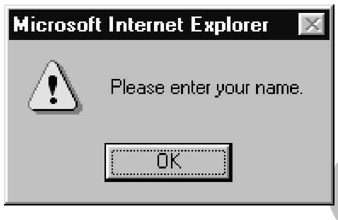
6. **Editor:** If the form element is either null or empty, create an alert dialog box that asks the user to enter information in the appropriate form element. *(Hint: Use the `name` property of each element in the argument for the `alert()` method.)* Then use the `focus()` method to place the user's cursor in the appropriate form element. Finally, create a `return false` statement. This statement will be returned to the function invocation statement and will cancel submission of the form. Close the `if` statement. Close the `for` loop.

7. **Editor:** After the `for` loop, create a `return true` statement. This statement will execute only if all form elements pass the validation test.

8. **Editor:** Another function named `emailTest()` has been created for you. This function will be discussed following this lab.

9. **Editor:** Save **optionalLab7-1.htm**.

10. **Browser:** Open **optionalLab7-1.htm**. Your screen should resemble Figure OL7-1. Click the **Submit Data** button without entering any data in the form. You should see an alert dialog box similar to Figure OL7-2. If you do not, verify that the source code you entered is correct.


Microsoft Internet Explorer

Please enter your name.

Figure OL7-2: Alert dialog box

11. **Browser:** After closing the alert dialog box, your cursor should be in the Name text box. If it is not, verify that the source code you entered is correct.

12. **Browser:** Experiment with the form to ensure that the proper alert dialog boxes appear for each situation. Ensure that the user's cursor is in the proper text box if the text box is left empty.

Optional Lab 8-1: Using passwords with cookies

In this lab, you will use a cookie that allows access to a second HTML page. If the cookie is present in the password-protected page and the user enters the correct password value, the user will be allowed to view the page. If the password is incorrect, the user will be returned to the previous page.

1. **Editor:** Open the **optionalLab8-1.htm** file from the Lesson_8 folder of the Student_Files directory.

2. **Editor:** Scroll down in the source code and examine the following code:

```
<FORM NAME="myForm">
<INPUT TYPE="text" NAME="pWord"><P>
<INPUT TYPE="button" VALUE="Submit" onClick="storePass(this.form);">
<INPUT TYPE="reset">
</FORM>
```

3. **Editor:** Locate the existing <SCRIPT> tags in the <HEAD> section of the document. Locate the comment that reads as follows:

```
// Create cookie here
```

Create a cookie named `password` and assign as its value the user's entry in the `pWord` text box.

4. **Editor:** The following code shows the `storePass()` function before your changes:

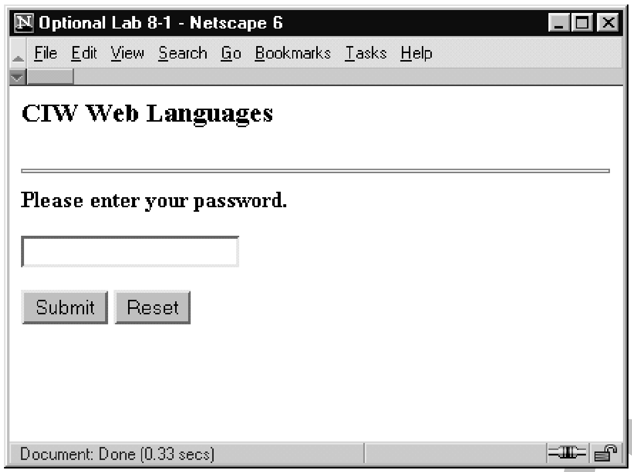
```
<SCRIPT LANGUAGE="JavaScript">
<!--
function storePass(form) {
   // Create cookie here


    alert(document.cookie);
    location.href = "pworded.htm";
}
//-->
</SCRIPT>
```

5. **Editor:** Save **optionalLab8-1.htm**.
6. **Editor:** Open **optionalLab8-1.1.htm** from the Lesson_8 folder of the Student_Files directory. Examine the following code in the <HEAD> section of the file:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
alert(document.cookie);
all = document.cookie;
if (all.indexOf("password=hello") != -1){
    alert("You have entered the correct password. Proceed.");
} else {
    alert("Incorrect password. Return and reenter.");
    location.href = "optionalLab8-1.htm";
}

//-->
</SCRIPT>
```

7. **Editor:** Close **optionalLab8-1.1.htm**.

8. **Browser:** Open **optionalLab8-1.htm**. Your screen should resemble Figure OL8-1.

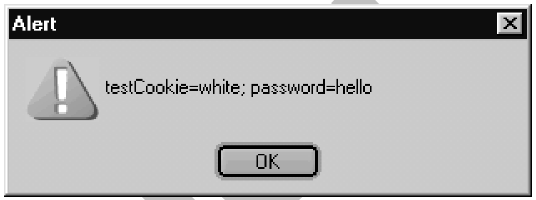

Optional Lab 8-1 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
Submit
Reset
Document: Done (0.33 secs)

CIW Web Languages

Please enter your password.

Figure OL8-1: OptionalLab8-1.htm

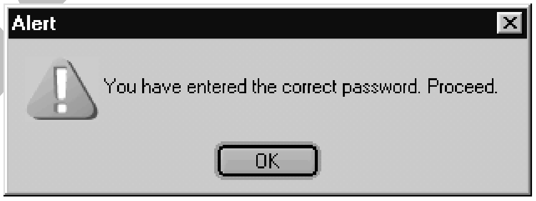
9. **Browser:** Enter the password **hello** and click the **Submit** button. This calls the `storePass()` function that stores your password into a cookie. Accept the cookie (if you are warned). You should then see the alert dialog box shown in Figure OL8-2.



testCookie=white; password=hello

Figure OL8-2: Alert dialog box

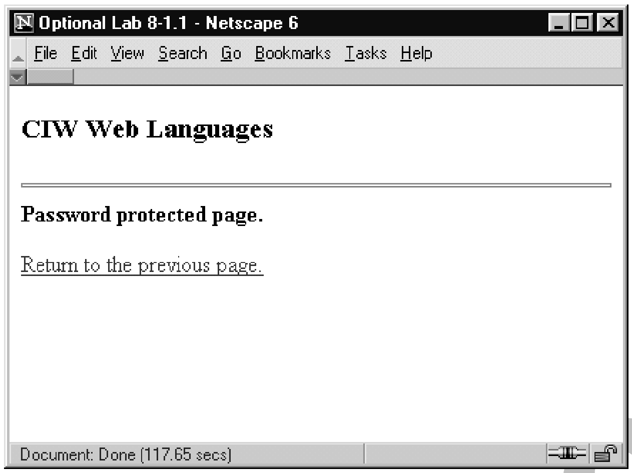
10. **Browser:** If you got a different message, you might have cleared the cookie assigned in the previous lab, in which case only the password *`name=value`* pair will appear. Click **OK** to continue. You should see another alert with the cookie's *`name=value`* pair, and then the alert dialog box shown in Figure OL8-3.



You have entered the correct password. Proceed.

Figure OL8-3: Alert dialog box

11. **Browser:** Click **OK** to continue. You will see the password-protected page, as shown in Figure OL8-4.


Optional Lab 8-1.1 - Netscape 6
File Edit View Search Go Bookmarks Tasks Help

CIW Web Languages

Password protected page.

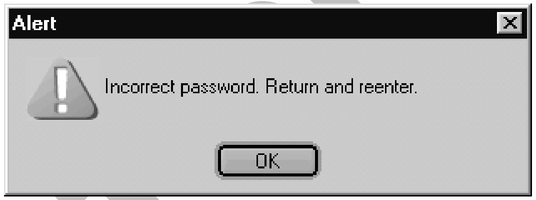
Return to the previous page.

Document: Done (117.65 secs)

Figure OL8-4: OptionalLab8-1.1.htm

12. Browser: To ensure that your script works for both correct and incorrect passwords, click the **Return** link to return to the previous page.

13. Browser: Enter an incorrect password and click the **Submit** button. After the alert displaying the cookie's *`name=value`* pair, you should see the message shown in Figure OL8-5.



Incorrect password. Return and reenter.

Figure OL8-5: Alert dialog box

14. Browser: Click **OK** to continue. Rather than seeing the file optionalLab8-1.1.htm, you should be returned to optionalLab8-1.htm.

This lab demonstrated another use of cookies. After you created the cookie named `password`, an alert displayed the cookie value to the user, and a `location.href` statement sent the user to the password-protected page. Following is the code from optionalLab8-1.1.htm:

```
alert(document.cookie);
all = document.cookie;
if (all.indexOf("password=hello") != -1){
    alert("You have entered the correct password. Proceed.");
} else {
    alert("Incorrect password. Return and reenter.");
    location.href = "optionalLab8-1.htm";
}
```

An alert displays the cookie's contents to the user. Next, a variable named `all` is assigned the value held in the cookie. An `if...else` statement then checks the string held in the variable `all` and sends the user back to password.htm if an incorrect password was entered. The most important line of code from this operation is the following:

```
if (all.indexOf("password=hello") != -1){
```

The `indexOf()` method is used to search the variable all for the string `password=hello`. If the integer returned by the `indexOf()` method is not `-1`, the cookie string must contain the string `password=hello`, meaning the user entered the correct password.

Note: This previous lab was presented for demonstration purposes only. Any sensitive content in a Web application should not be protected with the mechanism shown in this lab. It is much safer to store user names and passwords in a database. Server-side scripting in conjunction with the operating system's security features would then be used to authenticate and authorize users.

Optional Lab 9-1: Building a frameset page with JavaScript

In this optional lab, you will create a page that will allow users to check the price of CIW products and view an image of a product at the same time. This lab uses a simple frameset. The top frame will provide a drop-down menu from which the user can select a CIW product. You will create code that will write information to the bottom frame. The bottom frame will display an image of the product along with its name and price. Templates for this lab are provided in the OptionalLab_9-1 directory of the Lesson_9 folder in the Student_Files directory.

1. **Editor:** Open the **main.htm** file from the OptionalLab_9-1 folder. Examine the following code, then close the file:

```
<HTML>
<HEAD>
<TITLE>CIW Price Page</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
    function blankFrame() {
      return "<HTML><BODY> ➘
              <B>CIW items and prices</B></BODY></HTML>";
    }

    //-->
</SCRIPT>
</HEAD>
<FRAMESET ROWS="34%,*">
    <FRAME NAME="frame1" SRC="top.htm">
    <FRAME NAME="frame2" SRC="javascript:parent.blankFrame();">
</FRAMESET>
</HTML>
```

Note: This frameset uses a JavaScript function to supply a value for the SRC attribute of the second <FRAME> tag. The frame named `frame2` is the frame to which you will write.

2. **Editor:** Open **top.htm**. Locate the **`findPrice()`** function. The function begins with three variable declaration and initialization statements. Examine these statements:

```
var index = form.mySelect.selectedIndex;
var item = form.mySelect.options[index].text;
var price = form.mySelect.options[index].value;
```

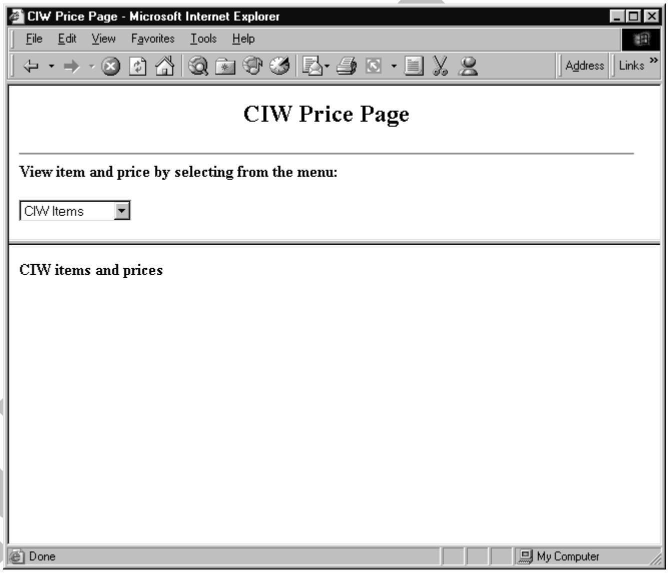
3. **Editor:** Following the variable statements, write code to open the data stream to the bottom frame. Then create a **`document.write`** statement to open <HTML>, <BODY> and <H3> tags. After the <H3> tag, concatenate the **`item`** variable into the string, then close the <H3> tag and add an <HR> tag. As shown previously, the `item` variable is assigned the `text` property from the `select` object.

4. **Editor:** Next, begin a **`switch`** statement using the `index` variable as the test expression. As shown previously, the `index` variable is assigned the integer returned from the `selectedIndex` property of the `select` object. For the `case 0` statement, add a **`break`** statement only. No action should occur if the user selects the default item from the drop-down menu.

5. **Editor:** For the `case 1` statement, add a **`document.write`** statement that will add an image to the bottom frame. For the `src` attribute, concatenate into the string the **`item`** variable (a corresponding .jpg file is found under each item's name in the

images directory). Remember to add the images directory to the `src` attribute's value. Concatenate the .jpg portion of the image's URL and then a `<BR>` tag after the `item` variable.

6. **Editor:** Still inside the `case 1` statement, add another **`document.write`** statement that will output the item's price underneath the image. Concatenate the `price` variable into this string. The **`price`** variable is assigned the item's price (the value of the `VALUE` attribute in the `select` object's `<OPTION>` tags).
7. **Editor:** Include two more **`case`** statements so that the three CIW products can be viewed. Note that each `case` statement is identical. Only the integer value following the `case` keyword will change.
8. **Editor:** Close the **`switch`** statement.
9. **Editor:** Add a **`document.write`** statement to close the **`<BODY>`** and **`<HTML>`** tags for the bottom frame. Close the data stream to the bottom frame.
10. **Editor:** Examine the rest of the file, then save **top.htm**.
11. **Browser:** Open **main.htm**. Your screen should resemble Figure OL9-1.


CIW Price Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
Done
My Computer

CIW Price Page

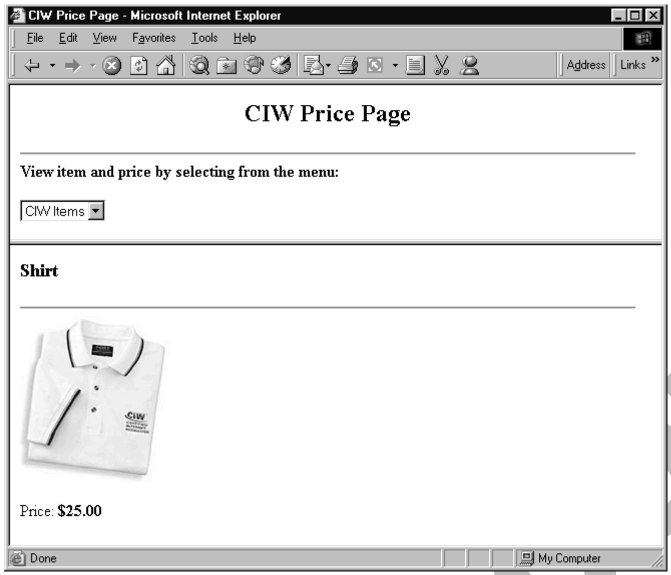
View item and price by selecting from the menu:

CIW Items

CIW items and prices

Figure OL9-1: Main.htm

12. **Browser:** Select the first item from the drop-down menu. Your screen should resemble Figure OL9-2. If it does not, open **top.htm** in your editor and check your work.


CIW Price Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
CIW Items
Done
My Computer

CIW Price Page

View item and price by selecting from the menu:

Shirt

Price: $25.00

Figure OL9-2: Main.htm with item selected

13. **Browser:** Select the other two items that you have scripted. The appropriate information should be displayed in the bottom frame.

Optional Lab 10-1: Creating a custom object

This lab will provide an opportunity for you to create and instantiate a custom object. You will then write code to display data pertaining to the custom object.

1. **Editor:** Open the **optionalLab10-1.htm** file from the Lesson_10 folder of the Student_Files directory.

2. **Editor:** Locate the comment that reads as follows:

   ```
   // Complete the employeeObject constructor
   ```

3. **Editor:** Complete the `employeeObject` constructor function that has been started for you. Add the properties that you see listed in the `employeeObject` constructor signature. Also add a method named `showEmployee`.

4. **Editor:** Locate the comment that reads as follows:

   ```
   // Instantiate 3 instances of employeeObject
   ```

5. **Editor:** Instantiate three instances of the `employeeObject`. Use the `employees` array that has been declared for you. Note that the first element of the `employees` array has already been defined. Make sure you start your array index numbers with `1` for the first employee. The values for the `employeeObject` properties have been provided for you.

6. **Editor:** Locate the comment that reads as follows:

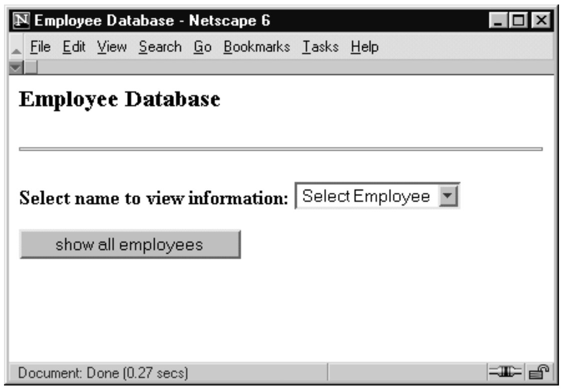
   ```
   // Complete the showEmployee() function
   ```

7. **Editor:** Complete the `showEmployees()` function. Use the `info` variable that has been declared for you. Use the += operator to build a string containing the text **Employee:**, **Department:** and **Extension:** with the appropriate `employeeObject` data concatenated in the appropriate locations. Concatenate a line break character after the name and department information.

8. **Editor:** An `alert()` method has been defined in the `showEmployees()` function that will display the `info` variable.

9. **Editor:** Examine the rest of the code, then save **optionalLab10-1.htm**.

10. **Browser:** Open **optionalLab10-1.htm**. Your screen should resemble Figure OL10-1.

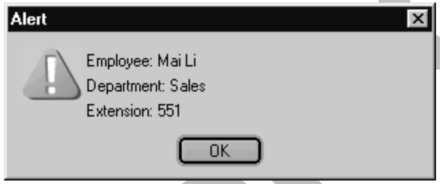

Employee Database - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
Select Employee
show all employees
Document: Done (0.27 secs)

Employee Database

Select name to view information:

Figure OL10-1: OptionalLab10-1.htm

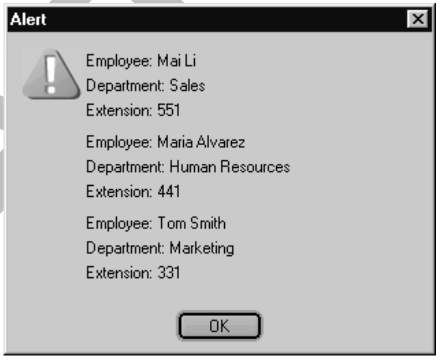
11. **Browser:** Select the first name from the drop-down menu. You should see an alert dialog box as shown in Figure OL10-2. If you do not, check the code you entered in optionalLab10-1.htm.



```
Employee: Mai Li
Department: Sales
Extension: 551
```

Figure OL10-2: Alert displaying employee information

12. **Browser:** Click the **show all employees** button. You should see an alert dialog box as shown in Figure OL10-3.



```
Employee: Mai Li
Department: Sales
Extension: 551

Employee: Maria Alvarez
Department: Human Resources
Extension: 441

Employee: Tom Smith
Department: Marketing
Extension: 331
```

Figure OL10-3: Alert showing all employees

13. **Browser:** Test all the names in the drop-down menu. Ensure that the proper information is displayed for each employee. If the proper data is not displayed, check the code you entered in optionalLab10-1.htm.

This lab offered you hands-on experience in creating, instantiating and displaying the data for a JavaScript custom object. You added code to a constructor function to make it a template for your custom objects. You added code to instantiate instances of your custom object, populating the object's properties with actual values. You also added code that extracted and displayed the data from your custom object.

This code in this lab used the `onChange` event handler to execute the `showEmployees()` function. Consider the following code.

```
<SELECT NAME="empName" onChange="employees[this.selectedIndex].showEmployee();
this.selectedIndex=0;">
```

As shown, the `selectedIndex` property of the `select` object is used as the subscript for the `employees` array. This determines the instance of the `employeeObject` that will invoke the `showEmployee()` function. As the value of the first element of the `employees` array is the text *Select Employee*, nothing should happen when that option is selected from the drop-down menu. The `selectedIndex` property of the `select` object is set back to `0` after each user selection. This action is in anticipation of the next user selection, and also ensures that the `onChange` event handler will never execute code if the first option is selected.

Though this lab created only three custom employee objects, hundreds of instances of the `employeeObject` could be instantiated, with little performance degradation. Remember that custom objects can provide a powerful client-side alternative to database servers or database applications when used in the right situations.

Handouts: Quizzes

Lesson 1 Quiz

1. What is a scripting language?

   a. Subset of a full programming language
   b. Meta-language used to create other languages
   c. Markup language used to format text and graphics
   d. Programming language used for dialog-based functions

2. The JavaScript language was developed by which of the following?

   a. Microsoft
   b. Sun Microsystems
   c. Netscape Corporation
   d. World Wide Web Consortium (W3C)

3. A key characteristic of JavaScript is that it is:

   a. platform-dependent.
   b. object-oriented.
   c. event-driven.
   d. difficult to learn.

4. Which of the following accurately describes a difference between JavaScript and Java?

   a. JavaScript is object-oriented whereas Java is object-based.
   b. JavaScript is interpreted by the client whereas Java is precompiled on the server.
   c. JavaScript uses static binding whereas Java uses dynamic binding.
   d. JavaScript requires strong typing whereas Java allows loose typing.

5. Which task is an example of a server-side JavaScript application?

   a. Working with frames
   b. Launching a new window from a Web page
   c. Validating fields of data in a user-submitted Web form
   d. Parsing and disseminating user-submitted information

6. Which task is an example of a client-side JavaScript application?

   a. Sending a message to a Web page user
   b. Saving client state in a multi-page process
   c. Enabling a server-side image map
   d. Connecting a Web page to a database

7. Which tag is used to embed JavaScript into an HTML document?

   a. The HTML `<LANGUAGE>` tag
   b. The JavaScript `<SCRIPT>` tag
   c. The HTML `<JAVASCRIPT>` tag
   d. The HTML `<SCRIPT>` tag

8. Which syntax indicates a multiple-line comment in JavaScript?

   a. `/*…*/`
   b. `<!--…-->`
   c. `//`
   d. `//-->`

Lesson 2 Quiz

1. Which method is used to request and capture data from a user?
   a. The `prompt()` method
   b. The `confirm()` method
   c. The `alert()` method
   d. The `open()` method

2. Which method is used to ask the user a yes-or-no type question?
   a. The `alert()` method
   b. The `open()` method
   c. The `confirm()` method
   d. The `prompt()` method

3. Which method is used to send a message to the user?
   a. The `open()` method
   b. The `alert()` method
   c. The `prompt()` method
   d. The `confirm()` method

4. In JavaScript, what is a keyword?
   a. A reference used to access a property or method of an object
   b. A word used to achieve specific results that cannot be used for any other purpose in JavaScript code
   c. Script within an HTML tag
   d. A named space of memory

5. In JavaScript, what is a reserved word?
   a. A container that holds a value
   b. A character used in an expression to store or return a value
   c. Part of a statement evaluated as a value
   d. A keyword intended for specified use that cannot be used for any other purpose in JavaScript code

6. In JavaScript, which of the following items allows you to store user-entered or programmer-defined data and use it more than once?
   a. Keyword
   b. Variable
   c. Operator
   d. Concatenation

7. In JavaScript, what is a data type?
   a. A container that holds a value
   b. A definition of the kind of information a variable holds
   c. A word used to achieve specific results
   d. A character used in an expression to store or return a value

8. In JavaScript, what is inline scripting?
   a. Part of a statement evaluated as a value
   b. A mechanism for associating code with an event
   c. Script within an HTML tag
   d. A reference used to access a property or method of an object

9. Which JavaScript variable data type can represent a series of alphanumeric characters?

   a. Object
   b. String
   c. Number
   d. Boolean

10. Which type of JavaScript expression evaluates to true or false?

    a. Assignment
    b. Arithmetic
    c. Boolean
    d. Logical

11. Which of the following examples is a valid variable name in JavaScript?

    a. `first&LastName`
    b. `1stName`
    c. `_lastName`
    d. `phone#`

12. Which statement uses proper JavaScript syntax for declaring variables?

    a. `var ~correctAnswer = "10";`
    b. `var "userName = Inga";`
    c. `var firstName = "Sal", lastName = "Fiore";`
    d. `var username == "Inga";`

13. Which JavaScript operator indicates concatenation?

    a. `=`
    b. `++`
    c. `+`
    d. `&&`

14. Which JavaScript operator indicates assignment?

    a. `=`
    b. `==`
    c. `!`
    d. `=>`

15. Which inline-scripted code will implement a simple event that takes place when the user first loads the Web page?

    a. `<BODY onUnload="alert('Hello!');">`
    b. `<SCRIPT LANGUAGE="JavaScript"> alert("Hello!"); </SCRIPT>`
    c. `<BODY onLoad="alert('Hello!');">`
    d. `<SCRIPT LANGUAGE="JavaScript"> onLoad="alert('Hello!')"; </SCRIPT>`

Lesson 3 Quiz

1. In JavaScript, what is a function?

   a. An expression containing data
   b. A named block of code that can be called as needed
   c. An action performed by an object that returns no value
   d. A statement that transfers program execution to a subroutine

2. The JavaScript `toUpperCase()` method is an example of:

   a. an argument.
   b. a calling statement.
   c. a built-in function.
   d. a user-defined function.

3. Which of the following examples is an example of a user-defined JavaScript function?

   a. `parseInt()`
   b. `function addSpace() {}`
   c. `isNaN()`
   d. `parseFloat()`

4. Which of the following examples is an example of a JavaScript conversion method?

   a. `document.write()`
   b. `parseFloat()`
   c. `alert()`
   d. `function addSpace()`

5. In JavaScript, what is an argument?

   a. Any method that returns a value
   b. A named block of code that can be called
   c. A value passed into a function from outside the function
   d. A statement that transfers program execution to a subroutine

6. In JavaScript, the `return` keyword is used to:

   a. return a value to a function's calling statement.
   b. return a value to a function.
   c. return a variable to an expression.
   d. return a function to an argument.

7. How does a global variable differ from local variable?

   a. You can access a global variable value from any function or `<SCRIPT>` block you define on the HTML page.
   b. You declare a local variable outside of any function.
   c. You declare a global variable within a function.
   d. You can access a local variable value from any function or `<SCRIPT>` block you define on the HTML page.

8. Which JavaScript event occurs when the user highlights the text in a form field?

   a. `blur`
   b. `select`
   c. `mouseOver`
   d. `click`

9. Which JavaScript event occurs when a Web page is accessed and appears in the browser?

   a. `submit`
   b. `focus`
   c. `change`
   d. `load`

10. Which JavaScript event handler can respond to the `checkbox` object?

    a. `onBlur`
    b. `onSubmit`
    c. `onMouseOver`
    d. `onSelect`

11. Which JavaScript event handler can respond to the `link` object?

    a. `onLoad`
    b. `onMouseOut`
    c. `onFocus`
    d. `onError`

12. Consider the following JavaScript statements:

```
function myTest (myValue) {
  if (myValue < 5) {
    return true;
  }else{
    return;
  }
}
document.write(myTest(6));
```

What is the result of these JavaScript statements? Why?

13. Consider the following JavaScript statements:

```
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!--

function myFunction() {
  myValue = 10;
  return myValue * 10;
}

//-->
</SCRIPT>

</HEAD>
<BODY>

<SCRIPT LANGUAGE="JavaScript">
<!--

document.write(myValue + "<BR>");
document.write(myFunction());

//-->
</SCRIPT>
</BODY>
```

What is the result of these JavaScript statements? Why?

Lesson 4 Quiz

1. Which JavaScript statement should you use to force the flow of control back to the top of a loop?

   a. `break`
   b. `continue`
   c. `switch`
   d. `do...while`

2. Which JavaScript statement should you use to execute a block of code for as long as a certain test condition is true, with the guarantee that the code block will execute at least once?

   a. `do`
   b. `do...while`
   c. `if`
   d. `if...else`

3. Which JavaScript statement should you use to compare a value against other values when searching for a match?

   a. `while`
   b. `if...else`
   c. `for`
   d. `switch`

4. Which JavaScript statement should you use to branch to one of two processes depending on the result of a test condition?

   a. `if...else`
   b. `for`
   c. `break`
   d. `do...while`

5. Which JavaScript statement should you use to execute a block of code for as long as a certain test condition is true?

   a. `if`
   b. `continue`
   c. `switch`
   d. `while`

6. Which JavaScript statement should you use to test for a single condition and branch to a process?

   a. `while`
   b. `if...else`
   c. `for`
   d. `if`

7. Which JavaScript statement should you use to exit a loop that would otherwise continue to execute?

   a. `if`
   b. `continue`
   c. `break`
   d. `switch`

8. Which JavaScript statement should you use to repeat a group of statements for some particular range of values?

   a. `if...else`
   b. `continue`
   c. `for`
   d. `do...while`

9. Write an `if` statement that checks a variable named `grade` for a range of values between 70 and 80. If the value is in this range, output the following text: Your grade is a C.

10. Consider the following code:

```
function calculateAvg(a, b, c) {
  var x = parseInt(a);
  var y = parseInt(b);
  var z = parseInt(c);

  return (x + y + z) / 3;
}
```

Write an `if...else` statement that calls the `calculateAvg()` function, then outputs one of the following: The average is 20 or less or The average is greater than 20.

11. Write a `while` loop that outputs the even numbers between 10 and 20.

12. Write a `for` loop that outputs the following:

```
10
9
8
7
6
5
4
3
2
1
0
```

13. Write a `for` loop that adds 10 numbers received from user input. Offer the user a chance to exit the program using the `break` statement.

14. Write a JavaScript program that asks the user to guess a number between one and five, then uses a `switch` statement to evaluate the user's input. Use three as the number to be guessed. Use a `default` clause to catch input that is not in the required range.

Lesson 5 Quiz

1. The JavaScript object model divides commonly used objects into what groups?

   a. Accessible elements of an HTML page or XML document
   b. Browser objects, language objects and form field objects
   c. `window`, `document` and `form` objects
   d. Objects supported by Microsoft Internet Explorer, Netscape Navigator, or both

2. Which JavaScript object is the default object?

   a. The `Function` object
   b. The `JavaScript` object
   c. The `String` object
   d. The `window` object

3. Which JavaScript object allows you to evaluate and change URL information?

   a. The `navigator` object
   b. The `history` object
   c. The `location` object
   d. The `link` object

4. Which JavaScript object is a combination of the content and interface elements that constitute the current Web page?

   a. The `frame` object
   b. The `document` object
   c. The `screen` object
   d. The `window` object

5. Which JavaScript object provides essential information if you must code your script for use in different browsers?

   a. The `navigator` object
   b. The `browser` object
   c. The `location` object
   d. The `history` object

6. Which JavaScript object allows you access to previously visited Web page URLs?

   a. The `location` object
   b. The `link` object
   c. The `navigator` object
   d. The `history` object

7. Which JavaScript object represents the frame of the browser and the mechanisms associated with it?

   a. The `document` object
   b. The `window` object
   c. The `frame` object
   d. The `navigator` object

8. Which JavaScript object can be used to animate a button whenever a mouse passes over it?
   a. The `button` object
   b. The `document` object
   c. The `applet` object
   d. The `image` object

9. Which of the following tools allows you to use several properties and/or methods with a single object?
   a. Dot notation
   b. The `with` statement
   c. The `write()` method
   d. The `navigator` object

10. Which general syntax should you use to open a new window with specific features using JavaScript?
    a. `open(URL,"Window Name",Feature List)`
    b. `window.open("URL","WindowName")`
    c. `open.window(URL,WindowName,Feature List)`
    d. `open("URL","WindowName","Feature List")`

11. Which property is associated with the `document` object?
    a. `title`
    b. `write()`
    c. `status`
    d. `frames`

12. Which property is commonly used with the `image` object?
    a. `open()`
    b. `location`
    c. `src`
    d. `onLoad`

13. Write a JavaScript statement that opens a new window that has scrollbars, menu, location, and is resizable. The new window's document will be dynamically created. Also, open the data stream to the new window's document.

14. What would be the size of the window opened in the previous question?

15. Write JavaScript code that sets the background and foreground colors of a document.

16. Write a JavaScript statement that outputs information concerning the most recent modification of a document.

17. Write a JavaScript code block that outputs the `src` property of the various images that an HTML page might contain.

18. Write JavaScript code to preload an image named myImage.jpg that exists in the images directory of the root directory.

19. A document has an image named `img1`. Create a button object that uses the `onClick` event handler to dynamically change img1 to the image that was preloaded in the previous question.

20. Write JavaScript statements that output the protocol used to access the current HTML page.

21. Write JavaScript statements that use a confirm dialog box and the `history` object to ask users if they want to go back two pages.

22. Write a line of JavaScript code that outputs the user's browser name and version.

Lesson 6 Quiz

1. Which JavaScript language object allows you to access and manipulate time information in your scripts?

   a. The `Date` object
   b. The `Math` object
   c. The `Array` object
   d. The `String` object

2. Which JavaScript language object is a static object that does not hold a value but contains various constants as properties?

   a. The `RegExp` object
   b. The `Date` object
   c. The `Array` object
   d. The `Math` object

3. Which JavaScript language object allows you search for specified patterns in text??

   a. The `Math` object
   b. The `Array` object
   c. The `RegExp` object
   d. The `String` object

4. Which JavaScript language object allows you to give a single variable multiple "slots," each with a different value, which can be referenced by index numbers?

   a. The `Array` object
   b. The `Math` object
   c. The `RegExp` object
   d. The `Date` object

5. Which JavaScript language object offers predefined methods for formatting text?

   a. The `Math` object
   b. The `Array` object
   c. The `RegExp` object
   d. The `String` object

6. Which JavaScript language object can be used to generate random quotations from a predefined array of quotes?

   a. The `Array` object
   b. The `String` object
   c. The `Math` object
   d. The `RegExp` object

7. Which `String` object method can be used to test user input?

   a. The `toLowCase()` method
   b. The `test()` method
   c. The `indexOf()` method
   d. The `link()` method

8. Which `RegExp` object method can return the index position of a sought-after string?

   a. The `search()` method
   b. The `test()` method
   c. The `indexOf()` method
   d. The `match()` method

9. Which `Array` object method can be used to create a string of an array's values?

   a. The `lastIndexOf()` method
   b. The `reverse()` method
   c. The `sort()` method
   d. The `join()` method

10. Which `Date` object method can be used to return the day of the week as a numeric value (0 through 6, where 0 is Sunday)?

    a. The `setDate()` method
    b. The `getDay()` method
    c. The `getDate()` method
    d. The `setDay()` method

11. Consider the following JavaScript statements:

```
var myStr = "Caught In The Wind";
document.write(myStr.charAt(0));
document.write(myStr.charAt(7));
document.write(myStr.charAt(14));
```

    What is the output of these statements?

    ____

12. Consider the following JavaScript statements:

```
var myStr = "joseB@company.com";
document.write(myStr.indexOf("@", 0) + "<BR>");
document.write(myStr.lastIndexOf("com") + "<BR>");
document.write(myStr.indexOf("b@", 0));
```

    What is the output of these statements?

    ____

    ____

    ____

13. Consider the following `searchString()` function:

```
function searchString(oneChar, testString) {
  var returnValue = false
  for (var i = 1; i <= testString.length; i++) {
    if (testString.substring(i, i + 1) == oneChar) {
      returnValue = true;
      break;
    }
  }
  return returnValue;
}
```

14. Consider the following array:

```
var myArray = new Array("red, "white", "blue);
```

Write code that outputs the values from this array on separate lines of an HTML page.

15. Consider the following JavaScript statements:

```
var myArray = new Array("red", "white", "blue");
document.write(myArray);
```

What is the output of these statements?

16. Consider the following JavaScript statements:

```
var myArray = new Array("red", 10, "white", 30, "blue", 5);
document.write(myArray.sort());
```

What is the output of these statements?

17. Write a JavaScript block of code that uses the `Date` object to send the user to an HTML page titled specifically for the day of the week.

18. Write a JavaScript block of code that returns the current hour and displays it to the user.

19. Write a JavaScript function that tests two numbers for the higher and outputs that number to the user.

Lesson 7 Quiz

1. Which example shows proper JavaScript syntax for referring to a `form` object?

   a. `document.newForm.fieldName.value;`
   b. `window.newForm.fieldName.value;`
   c. `form.newForm.fieldName."value"`
   d. `document.newForm[fieldName].value;`

2. In JavaScript, what is the purpose of the `form` object?

   a. To enclose the HTML `<FORM>` tags
   b. To represent all the forms in a single Web site
   c. To access all properties of a form
   d. To replace the HTML `<FORM>` tags

3. Which JavaScript object displays a single line of alphanumeric values and provides programmatic access to these user-entered values?

   a. The `textarea` object
   b. The `submit` object
   c. The `select` object
   d. The `text` object

4. Which JavaScript object allows you to choose one or more items from a list?

   a. The `select` object
   b. The `checkbox` object
   c. The `radio` object
   d. The `textarea` object

5. Which JavaScript input object can a user click to return a Boolean value of true or false?

   a. The `reset` object
   b. The `checkbox` object
   c. The `button` object
   d. The `select` object

6. Which JavaScript object provides a basic push-button interface and relies on scripting for its functionality?

   a. The `form` object
   b. The `submit` object
   c. The `button` object
   d. The `select` object

7. Which JavaScript object is used to select an option from among mutually exclusive options?

   a. The `select` object
   b. The `button` object
   c. The `checkbox` object
   d. The `radio` object

8. Which of the following is an event handler used by the `form` object?

   a. `onReset`
   b. `submit()`
   c. `onClick`
   d. `select()`

9. Which attribute defines a group of radio buttons for the `radio` object?

   a. `checked`
   b. `NAME`
   c. `TYPE`
   d. `value`

10. Which attribute is used to allow the user to choose more than one choice from a `select` object?

    a. `options`
    b. `SELECT`
    c. `length`
    d. `MULTIPLE`

11. Which of the following best describes the main purpose of client-side form validation with JavaScript?

    a. To verify that all user-entered data is true and correct
    b. To correct typing errors in user-entered data before processing
    c. To detect whether form elements hold no or a null value or properly formatted values
    d. To impress users with intervention from the program

12. Write a `<FORM>` tag that uses inline scripting to call a function named `clear()` when the form is reset.

    ____

13. Write a JavaScript function that writes true or false on a `button` object (named `myButton` in a form named `myForm`) depending on the result of a confirm dialog box.

14. Write JavaScript code to create a function that checks a `text` object (named `myText` in a form named `myForm`) for an empty value. If there is no value, ask the user to enter a value and place the cursor back in the `text` object.

15. Write JavaScript and HTML code to create a `text` object (named `myText` in a form named `myForm`) that contains the read-only string CIW Certified.

16. Write JavaScript and HTML code to create a `checkbox` object that, when clicked, displays its state (checked or not checked) and value in an alert dialog box.

17. Consider the following code:

```
<FORM NAME="myForm">
value1<INPUT TYPE="radio" NAME="myRadio" VALUE="test1">
value2<INPUT TYPE="radio" NAME="myRadio" VALUE="test2">
value3<INPUT TYPE="radio" NAME="myRadio" VALUE="test3">
</FORM>
```

Write a function that determines which `radio` object was selected, with the selected `radio` object's value reflected back to the user in an alert dialog box.

18. Consider the following code:

```
<FORM NAME="myForm">
<SELECT NAME="mySelect">
<OPTION VALUE="value1">value1
<OPTION VALUE="value2">value2
<OPTION VALUE="value3">value3
</SELECT>
</FORM>
```

Write a JavaScript function that determines which `select` object's option was chosen, with the chosen `select` option's value reflected back to the user in an alert dialog box.

Lesson 8 Quiz

1. A cookie can be scripted to:

   a. Track user activities on a server.
   b. Obtain files from a user's hard drive.
   c. Store information on a user's hard drive.
   d. Obtain user information from a Web site.

2. Which cookie parameter is required?

   a. `name=value`
   b. `expires=date`
   c. `path=path`
   d. `secure`

3. Which example demonstrates the general JavaScript syntax to assign a cookie?

   a. `window.cookie = "name = value";`
   b. `document.cookie = "name = value";`
   c. `cookie = "expires = date";`
   d. `prompt(document.cookie);`

4. Which example demonstrates proper JavaScript syntax to test for the presence of a cookie?

   a. `document.cookie = "name = value";`
   b. `confirm(cookie);`
   c. `alert(document.cookie);`
   d. `window.cookie = "name = value";`

5. You have just reassigned a cookie with an expiration date that has already passed. What will occur?

   a. The cookie will expire and generate a same-named replacement.
   b. The cookie will expire and be cleared.
   c. The cookie will send a request for update to its server.
   d. Nothing. You cannot reassign a cookie in this way.

6. How can a user control cookies in his or her browser?

   a. Adjust browser settings to choose which cookies to accept.
   b. Adjust browser settings to avoid Web sites that set cookies.
   c. Use Netscape Navigator because it does not accept cookies.
   d. Use Microsoft Internet Explorer because it does not accept cookies.

7. What is a signed script?

   a. An electronic mechanism that establishes identity
   b. JavaScript code that has been copyrighted by the author
   c. JavaScript code that allows you to create a digital certificate
   d. A program that can perform operations once considered a security risk

8. How does a cookie appear in an HTTP response header?

   ____

9. How would two cookies appear in a client's HTTP request header?

   ____

10. Write JavaScript code to create a function that sets a cookie named `user` that uses the user's input from the following form field as its value:

    ```
    <FORM NAME="myForm">
    ```

    Enter Name:

11. Consider the following malicious JavaScript code:

    ```
    var myColors = new Array;
    myColors[0] = "#FF0000";
    myColors[1] = "#00FF00";
    myColors[2] = "#0000FF";

    for(var i = 0; i < myColors.length; i++){
      document.bgColor = myColors[i];
        if(i == 2){
          i = 0;
        }
    }
    ```

Lesson 9 Quiz

1. To target a frame in JavaScript, you need to know:

   a. Its parent frame reference.
   b. Its location in the embedded frameset.
   c. Both its name and its number in the `frames` array.
   d. Either its name or its number in the `frames` array.

2. Which JavaScript keyword targets the parent of all parent files in a frameset?

   a. `_top`
   b. `_parent`
   c. `top`
   d. `location`

3. To change two frames simultaneously, you must write a JavaScript function that includes what?

   a. The `parent.parent` statement
   b. Two `opener` statements
   c. Two references to `top`
   d. Two `location.href` statements

4. Which example shows the appropriate general reference syntax for accessing a function stored in a frame file?

   a. `frameName.functionName();`
   b. `relationship.frameName.functionName();`
   c. `relationship.frameName.variableName;`
   d. `frameName.opener.functionName;`

5. Which example shows the appropriate general reference syntax for accessing a variable defined on a parent window from the child window?

   a. `parent.windowName.variableName();`
   b. `child.frameName.functionName();`
   c. `window.opener.variableName;`
   d. `windowName. variableName;`

6. Which of the following allows you to access the methods and properties of the parent window from a newly opened window?

   a. The `location.href` statement
   b. The `open()` method of the `window` object
   c. The `opener` property of the `window` object
   d. The `parent.parent` statement

7. Which of the following allows you to access a child window from the parent window using an object reference?

   a. The `opener` property of the `window` object
   b. The `parent.parent` statement
   c. The `location.href` statement
   d. The `open()` method of the `window` object

8. A function named `myFunction()` is defined on a master frameset page. The frameset uses an embedded frameset defined in a file named 2Frame. The minor.htm file defines two frames named `bFrame` and `cFrame`. Following are the definitions of the frameset pages:

```
<!-- Main frameset -->
<HTML>
<HEAD>
<TITLE>Main Frameset</TITLE>
<SCRIPT LANGUAGE="JavaScript">
  function myFunction() {
    alert("myFunction() called.");
  }
</SCRIPT>
</HEAD>

<FRAMESET COLS="15%,*">
<FRAME NAME="toc" SRC="toc.htm">
<FRAME NAME="2Frame" SRC="minor.htm">
</FRAMESET>
</HTML>

<!-- Second frameset -->
<HTML>
<HEAD>
<TITLE>2Frame</TITLE>
</HEAD>

<FRAMESET rows="20%,*">
<FRAME NAME="bFrame" SRC="b.htm">
<FRAME NAME="cFrame" SRC="c.htm">
</FRAMESET>
</HTML>
```

How would you invoke `myFunction()` from the frame named `cFrame`?

9. Consider the following JavaScript statement:

```
open("new.htm","newWin","height=300,width=300");
```

How would you change the background color of the document in the new window opened with this statement?

10. Consider the following `myWin()` function:

```
var myWin;
function myFunction() {
  alert("myFunction() called.");
  myWin.focus();
}

function myWin(x) {
  myWin = open("","","height=200,width=200");

}
```

Complete the `myWin()` function. The function will open a data stream to the new window. Add a `button` object to the new window that will invoke `myFunction()` when clicked.

Lesson 10 Quiz

1. What is the first step in creating a custom object?
   a. Creating a method
   b. Defining a constructor
   c. Naming property values
   d. Creating a property

2. What is a constructor?
   a. A new instance of a custom object
   b. A method that defines the properties of a custom object
   c. A predefined object to which you add methods and properties
   d. A function that defines methods and properties of a custom object used as a template for instances of the custom object

3. What is instantiation?
   a. The process of defining a constructor
   b. The process of creating new copies of an object
   c. The process of defining properties and methods of a custom object
   d. The process of returning specific method parameters to the calling statement

4. How do you instantiate new instances of an object?
   a. With the `new` keyword, the function name, and a list of values
   b. With the `new` keyword and the object name
   c. With the `this` keyword and the object name
   d. With the `this` keyword, the function name, and a list of parameters

5. How do you access the properties and methods of a newly instantiated object?
   a. With the `this` keyword
   b. By declaring variables
   c. Using dot notation
   d. With the `new` keyword

6. Which of the following operates on single instances of custom objects rather than on all objects?
   a. Methods
   b. Functions
   c. Properties
   d. Arrays

7. Define a constructor function for a custom object named `empObject`. Add three properties: `name`, `age`, and `department`. Add one method: `showOne`

8. Using the `empObject()` constructor function from the previous question, instantiate two new employees in an array named `employees`.

9. Using the first employee instantiated in the previous question, how would a program access the `name` and `age` properties for that object? Output the data to two separate lines on an HTML page.

10. Using the `employees` array, write a script block that would output each employee's name while making each name a link to the `showOne()` method. You will define the `showOne()` method in the next question.

11. Consider the following `showOne()` function:

```
function showOne() {
}
```

Complete the `showOne()` function so that an alert dialog box gives the user the appropriate information for each employee. Remember that `showOne()` is a method of the `empObject()` constructor and will be called for one specific instance of the custom object.

12. Consider the following form:

```
<FORM NAME="myForm">
<INPUT TYPE="text" NAME="name">
<BR>
<INPUT TYPE="button" VALUE="Get Department"
onClick="getDepartment(this.form);">
</FORM>

function getDepartment(form) {

}
```

Using this form, complete the `getDepartment()` function so that users can enter a name in the text box and receive an alert dialog box informing them of the employee's department. Add functionality that informs users if a match is not found for the entered name.

13. Suppose you need a function named `getAllEmployees()` that returns all instances of employee objects. Using the `empObject` from a previous question, write this function. Use `document.write()` statements to output the data to an HTML page.

Handout: Course Assessment

The following multiple-choice post-course assessment will evaluate your knowledge of the skills and concepts taught in JavaScript Fundamentals.

1. Which of the following code segments properly ensures that JavaScript code will not be displayed in the browser window by older browsers?

   a.
   ```
   <!--
   <SCRIPT LANGUAGE="JavaScript">
   // JavaScript code
     var x = 10;
   </SCRIPT>
   -->
   ```

   b.
   ```
   <SCRIPT LANGUAGE="JavaScript">
   <!--
   // JavaScript code
     var x = 10;
   -->
   </SCRIPT>
   ```

   c.
   ```
   <!--
   <SCRIPT LANGUAGE="JavaScript">
   // JavaScript code
     var x = 10;
   </SCRIPT>
   //-->
   ```

   d.
   ```
   <SCRIPT LANGUAGE="JavaScript">
   <!--
   // JavaScript code
     var x = 10;
   //-->
   </SCRIPT>
   ```

2. Which of the following code segments properly specifies JavaScript for an HTML 4.0-compliant document?

   a. `<SCRIPT TYPE="text/javascript">`
   b. `<SCRIPT TEXT="code/javascript">`
   c. `<SCRIPT TYPE="code/javascript">`
   d. `<SCRIPT LANGUAGE="text/javascript">`

3. Which of the following JavaScript code segments correctly concatenates the variables `x` and `y` into a `document.write()` statement?

   a.
   ```
   var x = 10, y = 20;
   document.write("x is " & x & " and y is " & y & ".");
   ```

   b.
   ```
   var x = 10, y = 20;
   document.write("x is " + x + " and y is " + y + ".");
   ```

   c.
   ```
   var x = 10, y = 20;
   document.write(x is & " x " & and y is & " y " & .);
   ```

   d.
   ```
   var x = 10, y = 20;
   document.write("x is" + x + "and y is" + y + ".");
   ```

4. What is the return value when a user selects the Cancel button from a JavaScript confirm dialog box?

   a. true
   b. null
   c. false
   d. There is no return value.

5. A JavaScript developer wants to use a confirm dialog box to verify that a user indeed meant to click a particular link. Which of the following code segments properly scripts such an <A> tag?

   a. `<A HREF="http://www.ciwcertified.com/" onClick="confirm("Proceed?");">Visit CIW</A>`

   b. `<A HREF="http://www.ciwcertified.com/" onClick="return confirm('Proceed?');">Visit CIW</A>`

   c. `<A HREF="http://www.ciwcertified.com/" onClick="return confirm("Proceed?");">Visit CIW</A>`

   d. `<A HREF="http://www.ciwcertified.com/" onClick="confirm('Proceed?');">Visit CIW</A>`

6. What is the output of the following JavaScript code segment?

```
var x = 30,  y = 20;

document.write(++x * y--);
```

   a. 589
   b. 570
   c. 651
   d. 620

7. Which of the following JavaScript code segments properly calls and passes arguments to a function named myFunction()?

   a. `call myFunction(text  more text);`
   b. `myFunction("text", "more text");`
   c. `invoke myFunction(text, more text);`
   d. `myFunction("text"; "more text");`

8. Which of the following JavaScript event handlers is invoked when a cursor is placed in a `text` object?

   a. `onSelect`
   b. `onChange`
   c. `onClick`
   d. `onFocus`

9. Which of the following code segments properly scripts a JavaScript `for` statement?

   a.
```
for (var i = 0, i <= 10, i--) {
  document.write("loop number " + i + "<BR>");
}
```

   b.
```
for (var i = 0; i >= 10; i++) {
  document.write("loop number " + i + "<BR>");
}
```

   c.
```
for (var i = 0; i <= 10; i++) {
  document.write("loop number " + i + "<BR>");
}
```

   d.
```
for (var i = 0, i >= 0, i--) {
  document.write("loop number " + i + "<BR>");
}
```

10. Which of the following code segments correctly scripts a JavaScript `do...while` statement?

   a.
```
var i = 1;
do {
  document.write("loop number " + i + "<BR>");
  i--;
while(i < 10);
}
```

   b.
```
var i = 1;
do {
  document.write("loop number " + i + "<BR>");
  i++;
} while(i < 10);
```

   c.
```
var i = 1;
do {
  document.write("loop number " + i + "<BR>");
  i++;
while(i < 10)
}
```

   d.
```
var i = 1;
do {
  document.write("loop number " + i + "<BR>");
} while(i < 10);
```

11. Which of the following JavaScript code segments correctly opens a new window?

    a.
    ```
    var myWindow
    myWindow =
    window.open("newWin.htm";"myWin";"scrollbars=1,menu=1");
    ```

    b.
    ```
    var myWindow
    myWindow = window.open("newWin.htm","my Win",
    scrollbar=1,menus=1");
    ```

    c.
    ```
    var myWindow
    myWindow =
    window.open("newWin.htm";"myWin";scrollbars=1;menus=1");
    ```

    d.
    ```
    var myWindow
    myWindow =
    window.open("newWin.htm","myWin",scrollbars=1,menu=1");
    ```

12. Which of the following properties returns the date and time that a `document` object was last revised?

    a. `lastModified`
    b. `lastChanged`
    c. `lastModification`
    d. `lastAltered`

13. Which of the following code segments correctly uses JavaScript to preload images for an HTML page?

    a.
```
if (document.images) {

    var pic1, pic2;

    pic1 = new image();
    pic1.source = "images/pic1.gif";

    pic2 = new image();
    pic2.source = "images/pic2.gif";
  }
```

    b.
```
if (document.images) {

    var pic1, pic2;

    pic1 = new Image();
    pic1.src = "images/pic1.gif";

    pic2 = new Image();
    pic2.src = "images/pic2.gif";
  }
```

    c.
```
if (document.images) {

    var pic1, pic2;

    pic1 = new Image();
    pic1.source = "images/pic1.gif";

    pic2 = new Image();
    pic2.source = "images/pic2.gif";
  }
```

    d.
```
if (document.Images) {

    var pic1, pic2;

    pic1 = new image();
    pic1.src = "images/pic1.gif";

    pic2 = new image();
    pic2.src = "images/pic2.gif";
  }
```

14. Which of the following JavaScript code segments correctly applies `String` object formatting methods to a string?

    a.
    ```
    var aString = "test text.";
    aString.toUpperCase();
    aString.italics();
    document.write(aString);
    ```

    b.
    ```
    var aString = "test text.";
    aString = aString.toUpperCase().italics();
    document.write(aString);
    ```

    c.
    ```
    var aString = "test text.";
    aString.toUpperCase(aString);
    aString.italics(aString);
    document.write(aString);
    ```

    d.
    ```
    var aString = "test text.";
    aString.toUpperCase(aString).italic(aString);
    document.write(aString);
    ```

15. Which of the following JavaScript code segments correctly accesses the user's entry in a `text` object named `myTextBox` in a form named `myForm`?

    a.
    ```
    var myVar;
    myVar = document.myForm.myTextbox.text;
    ```

    b.
    ```
    var myVar;
    myVar = document.myForm.myTextbox.input;
    ```

    c.
    ```
    var myVar;
    myVar = document.myForm.myTextbox.value;
    ```

    d.
    ```
    var myVar;
    myVar = document.myForm.myTextbox.content;
    ```

16. Which of the following JavaScript code segments properly assigns a Boolean value to the `myVar` variable indicating the state of a `checkbox` object? Assume that the `checkbox` object is the second form element defined in the first form of an HTML document.

    a.
    ```
    var myVar;
    myVar = document.forms[1].elements[2].selected;
    ```

    b.
    ```
    var myVar;
    myVar = document.forms[0].elements[1].checked;
    ```

    c.
    ```
    var myar;
    myVar = document.forms[0].elements[1].selected;
    ```

    d.
    ```
    var myar;
    myVar = document.forms[1].elements[2].checked;
    ```

17. Which of the following JavaScript code segments properly accesses the numeric value returned when a user selects an option from a `select` object named `mySelect` in a form named `myForm`?

    a.
    ```
    var myar;
    myVar = document.myForm.mySelect.value;
    ```

    b.
    ```
    var myar;
    myVar = document.myForm.mySelect.selectedInt;
    ```

    c.
    ```
    var myar;
    myVar = document.myForm.mySelect.integer;
    ```

    d.
    ```
    var myar;
    myVar = document.myForm.mySelect.selectedIndex;
    ```

18. Which of the following JavaScript code segments correctly determines which radio button has been selected in a radio button group named `myRadio` in a form named `myForm`?

    a.
    ```
    var len = myRadio.length;
    var myVar = "";
    for(var idx = 0; idx < len; idx++) {
      if(document.myForm.myRadio[idx].selected) {
        myVar = document.myForm.myRadio.value;
      }
    }
    ```

    b.
    ```
    var len = myRadio.length;
    var myVar = "";
    for(var idx = 0; idx < len; idx++) {
      if(document.myForm.myRadio[idx].checked) {
        myVar = document.myForm.myRadio[idx].value;
      }
    }
    ```

    c.
    ```
    var len = myRadio.length;
    var myVar = "";
    for(var idx = 0; idx < len; idx++) {
      if(document.myForm.myRadio[idx].selected) {
        myVar = document.myForm.myRadio[idx].value;
      }
    }
    ```

    d.
    ```
    var len = myRadio.length;
    var myVar = "";
    for(var idx = 0; idx < len; idx++) {
      if(document.myForm.myRadio.checked) {
        myVar = document.myForm.myRadio[idx].value;
      }
    }
    ```

19. Which of the following JavaScript code segments indicates that it is not true that the variable `myVar` is zero or the variable `myVar2` is less than or equal to 20?

    a. `!(myVar == 0 || myVar2 =< 20);`
    b. `!(myVar != 0) && myVar2 <= 20;`
    c. `!(myVar == 0) || myVar2 <= 20;`
    d. `myVar != 0 && myVar2 =< 20;`

20. Which of the following code segments correctly creates and populates a JavaScript array?

    a.
    ```
    var myArray = new Array[];
    myArray(0) = "value 1";
    myArray(2) = "value 2";
    ```

    b.
    ```
    var myArray = new Array();
    myArray(0) = "value 1";
    myArray(2) = "value 2";
    ```

    c.
    ```
    var myArray = new Array[];
    myArray[0] = "value 1";
    myArray[2] = "value 2";
    ```

    d.
    ```
    var myArray = new Array();
    myArray[0] = "value 1";
    myArray[2] = "value 2";
    ```

21. What value does `myStr` hold after execution of the following JavaScript code segment?

    ```
    var myStr = "test string".substring(6, 2);
    ```

    a. st s
    b. tr
    c. est s
    d. st st

22. What value does `myVar` hold after execution of the following JavaScript code segment?

    ```
    var myVar = "test string".indexOf("st", 3);
    ```

    a. 3
    b. 5
    c. 6
    d. –1

23. What value does `myVar` hold after execution of the following JavaScript code segment?

    ```
    var myVar = "test string".indexOf("sts");
    ```

    a. -1
    b. 2
    c. 4
    d. 3

24. In general terms, what is the value of `myVar` after execution of the following JavaScript code segment?

    ```
    var myVar = navigator.appVersion.substring(0, 1);
    ```

    a. The first letter of the name of the browser.
    b. The second letter of the name of the browser.
    c. The version number of the browser.
    d. The first number in the version number of the browser.

www.CIWcertified.com

ISBN 1-58143-844-3

9 781581 438444