

JavaScript

Chapter 1
Introduction to JavaScript

Objectives

When you complete this chapter, you will be able to:

- Explain the history of the World Wide Web
- Describe the difference between client-side and server-side scripting
- Understand the components of a JavaScript statement
- Add basic JavaScript code to your web pages
- Structure your JavaScript programs

Introduction to the World Wide Web

- 1962: memos by J. C. R. Licklider
- 1960s: ARPANET
 - First Internet implementation
 - Developed by Advanced Research Projects Agency (ARPA): U.S. Department of Defense
- 1990 and 1991: World Wide Web (web)
 - Created by Tim Berners-Lee
 - Hypertext linking
 - Method of accessing cross-referenced documents

Introduction to the World Wide Web (cont'd.)

- “web” and “Internet”
 - Not synonymous
- Hypertext link or hyperlink
 - Reference to a specific web page: click to open page
- Web page
 - Document on the web
 - Identified by the Uniform Resource Locator (URL)
 - Unique Web address
- Uniform Resource Identifier (URI)
 - Many types of names and addresses on the web

Introduction to the World Wide Web (cont'd.)

- Website
 - Location on the Internet
 - Contains web pages and related files belonging to a company, organization, individual
- Web browser
 - Program displaying web page on the screen
- Web server
 - Computer delivering web pages
 - Handles requests
 - Provides responses

Understanding Web Browsers

- NCSA Mosaic
 - 1993: created at the University of Illinois
 - Allowed web navigation using a graphical user interface (GUI)
- Netscape Navigator
 - 1994: released by Netscape
 - Soon controlled 75% of the market
- Microsoft Internet Explorer
 - 1996: released by Microsoft

Understanding Web Browsers (cont'd.)

- Browser wars began over DHTML
 - Forced web industry to rapidly develop and adopt advanced web page standards
- 1994: World Wide Web Consortium (W3C) established
 - Oversee web technology standards development
 - Adopted Internet Explorer version four DHTML
 - Loyal Netscape followers defected to Microsoft
- Major modern browsers
 - Internet Explorer, Mozilla Firefox, Google Chrome

Creating Web Pages

- Hypertext Markup Language (HTML)
 - Markup language
 - Set of characters or symbols defining a document's logical structure
 - Based on an older Standard Generalized Markup Language (SGML)

Basic HTML Syntax

- Tags: formatting instructions
 - Specify how data in document is treated by browser
- Element: tag pair and any data it contains
 - Element content: information contained between element's opening and closing tags
 - Empty elements: do not require a closing tag
- Root element (`html`): contains all other elements in a document
- `<head>` element: information used by the browser
- Document body: `<body>` element and the text and elements it contains

Basic HTML Syntax (cont'd.)

HTML ELEMENT NAME	DESCRIPTION
article	Marks the main content of a web document
body	Marks the body of an HTML document
div	Marks a generic section of the web page body
head	Marks the page header and contains information about the entire page
h n	Marks heading level elements, where n represents a number from 1 to 6
html	Marks the content of an HTML document
img	Inserts an image file
nav	Marks navigation options, such as a navigation bar at the top or bottom of a page or along its side
p	Identifies the marked text as a paragraph

Table 1-1 Common HTML elements

Basic HTML Syntax (cont'd.)

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="utf-8" />
5          <title>Hotel Natoma - Reservations</title>
6          <link type="text/css" rel="stylesheet" media="screen"
7              href="natoma.css" />
8          <link type="text/css" rel="stylesheet" media="screen"
9              href="hnform.css" />
10         <link rel="shortcut icon" href="favicon.ico" />
11     </head>
12     <body>
13         <div id="box">
14             <h1>
15                 
17             </h1>
18             <nav>
19                 <ul id="mainnav">
20                     <li><a href="index.html">Home</a></li>
21                     <li><a href="nearby.html">
22                         What's Nearby</a></li>
23                     <li><a href="http://bit.ly/bb3Sic"
24                         target="_blank">Location</a></li>
25                     <li><a href="museums.html">SF Museums</a></li>
26                     <li><a href="greensf.html">Green SF</a></li>
27                     <li><a href="reserve.html">Reservations</a></li>
28                 </ul>
29             </nav>
30             <article>
31                 <h2 id="main">Reservations</h2>
32             ...
```



Figure 1-1 Web page in a browser

Creating an HTML Document

- Text editors: Notepad or TextEdit
 - Word-processing applications capable of creating simple text files
- Web development tools: Adobe Dreamweaver and Microsoft Visual Studio
 - Graphical interfaces allowing immediate view of the results
 - Automates process of applying elements
 - May add many unfamiliar elements and attributes to documents

Creating an HTML Document

- Text editors created for coding
 - Non-graphical
 - Number lines of code
 - Color code text based on meaning
- Many available for free:
 - Aptana Studio 3 (Windows and Mac)
 - Komodo Edit (Windows and Mac)
 - Notepad++ (Windows)
 - TextWrangler (Mac)

Working with HTML5

- HTML5
 - Most current version of HTML
- Extensible Hypertext Markup Language (XHTML)
 - Once seen as future language for web development
 - Aspects including inflexible syntax kept from being widely embraced

Introduction to Web Development

- Web page design (web design)
 - Visual design and creation of documents appearing on the World Wide Web
- Web page authoring (web authoring)
 - Creation and assembly of the tags, attributes, data making up a web page
- Web development or web programming
 - Design of software applications for a website
- Web browsers contain commands to view underlying HTML code
 - Only view to improve skills

Understanding Client/Server Architecture

- Two-tier system
 - Client and server
- Server or back end
 - Usually a database: client requests information
- Client or front end
 - Responsible for user interface
 - Gathers information from user
 - Submits information to server
 - Receives, formats, presents results returned from the server

Understanding Client/Server Architecture (cont'd.)

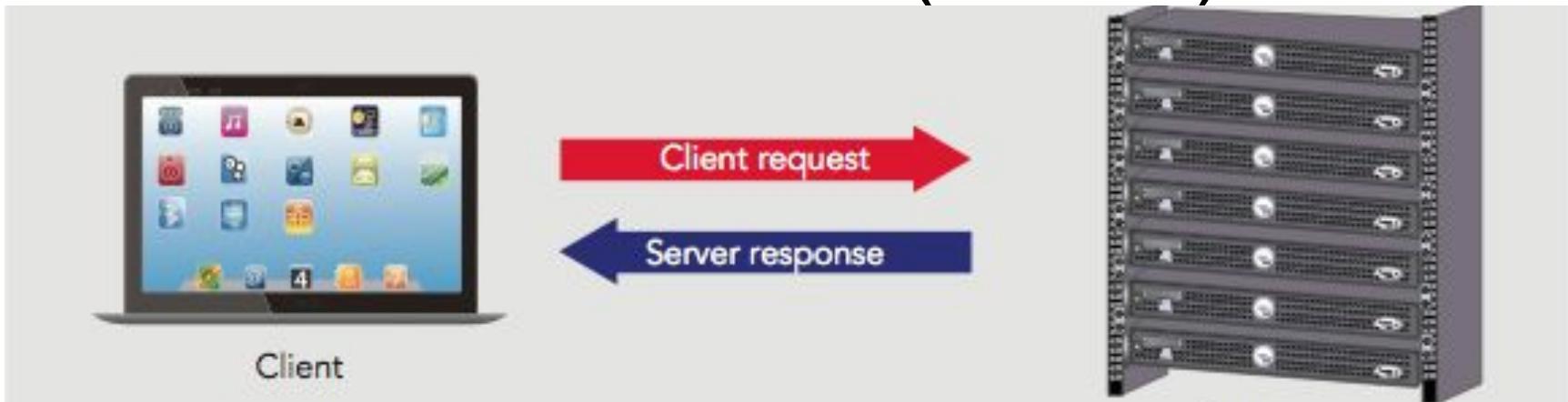


Figure 1-5 A two-tier client/server system

- Web built on a two-tier client/server system
 - Requests and responses through which a web browser and web server communicate happen with HTTP

Understanding Client/Server Architecture (cont'd.)

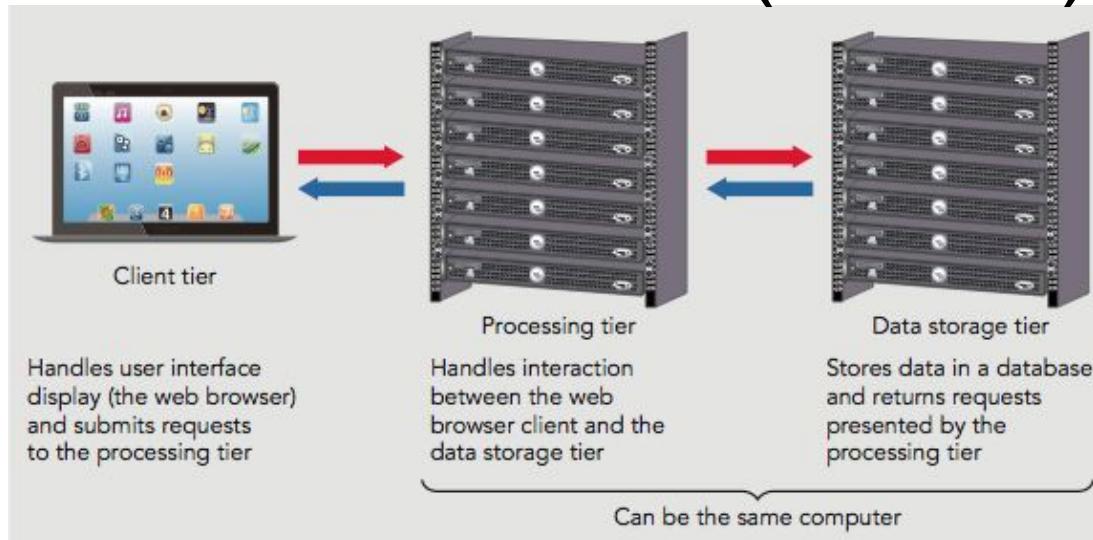


Figure 1-6 A three-tier client/server system

- Three-tier, multitier, client/server system
 - Client tier
 - Processing tier
 - Data storage tier

JavaScript and Client-Side Scripting

- Static web pages
 - Cannot change after browser renders them
- HTML produced static documents
- JavaScript
 - Allows web page authors to develop interactive web pages and sites
 - Client-side scripting language: runs on local browser
- Scripting engine executes scripting language code
- Scripting host
 - Web browser containing scripting engine

JavaScript and Client-Side Scripting (cont'd.)

- JavaScript history
 - First introduced in Navigator (LiveScript)
 - Navigator 2.0: name changed to JavaScript 1.0
 - Microsoft released Internet Explorer 4.0 version of JavaScript (Jscript)
- ECMA-Script
 - International, standardized version of JavaScript
 - Most recent version: edition 5.1

JavaScript and Client-Side Scripting (cont'd.)

- Limitations of JavaScript
 - Cannot be used outside the web browser
 - Cannot run system commands or execute programs on a client

Understanding Server-Side Scripting

- Server-side scripting
 - Scripting language executed from a web server
 - Popular languages: PHP, ASP.NET, Python, Ruby
- Can develop interactive web sites to communicate with a database
- Server-side scripting language limitations
 - Cannot access or manipulate a web browser
 - Cannot run on a client tier

Understanding Server-Side Scripting (cont'd.)

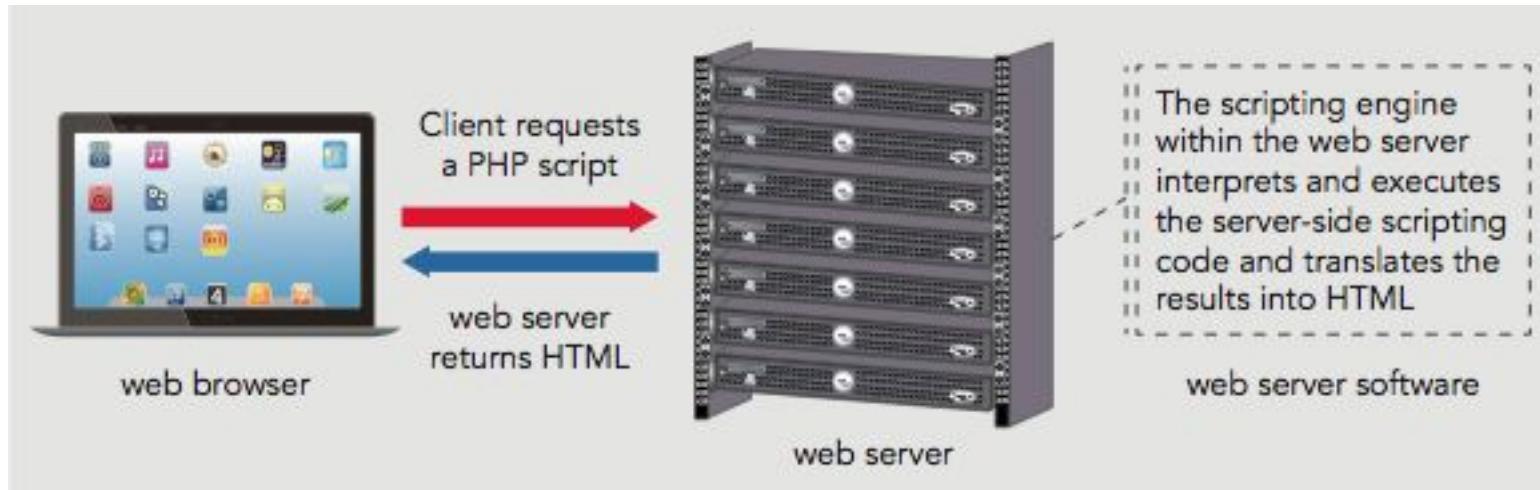


Figure 1-7 How a web server processes a server-side script

Should You Use Client-Side or Server-Side Scripting?

- General rule of thumb
 - Allow client to handle user interface processing and light processing (data validation)
 - Have the web server perform intensive calculations and data storage
- Important to perform as much processing as possible on the client

Adding JavaScript to Your Web Pages

- Basic procedures
 - Used for adding JavaScript to web pages

Using the script Element

- Scripts
 - JavaScript programs contained within a web page
- script element
 - Tells the browser that the scripting engine must interpret the commands it contains

Understanding JavaScript Objects

- Object
 - Programming code and data
 - Treated as an individual unit or component
- Procedures
 - Individual statements used in a computer program grouped into logical units
 - Used to perform specific tasks
- Methods
 - Procedures associated with an object

Understanding JavaScript Objects (cont'd.)

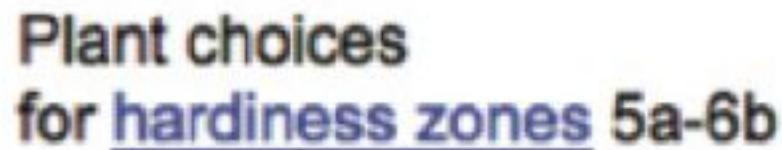
- Property
 - Piece of data associated with an object
 - Assign value to a property using an equal sign
- Argument
 - Information that must be provided to a method
- Passing arguments
 - Providing an argument for a method

Using the `write()` Method

- Document object
 - Represents content of a browser's window
- Create new web page text with the `write()` method of the Document object
 - Method requires a text string as an argument
 - Text string or literal string
 - Text contained within double or single quotation marks

Using the write() Method (cont'd.)

```
1 <script>
2   document.write("<p>Plant choices<br />");
3   document.write("for <a href=<br />
4     'http://planthardiness.ars.usda.gov'><br />
5       hardness zones</a> 5a-6b</p>");
```



Plant choices
for hardiness zones 5a-6b

Figure 1-8 Output of a script that uses the write() method of the Document object

Case Sensitivity in JavaScript

- JavaScript is case sensitive
- Within JavaScript code:
 - Object names must always be all lowercase

Adding Comments to a JavaScript Program

- Comments
 - Nonprinting lines placed in code containing various types of remarks
- Line comment
 - Hides a single line of code
 - Add two slashes // before the comment text
- Block comments
 - Hide multiple lines of code
 - Add /* before the first character included in the block and */ after the last character in the block

Writing Basic JavaScript Code

- Learn how to write basic JavaScript code
 - Start with variables

Using Variables

- **Variables**
 - Values a program stores in computer memory
- **Assigning a value to a variable**
 - Same as storing a value in a variable

Assigning Variable Names

- Identifier
 - Name assigned to a variable
 - Rules and conventions
 - Must begin with an uppercase or lowercase ASCII letter, dollar sign (\$), or underscore (_)
 - Can use numbers in an identifier: not as the first character
 - Cannot include spaces in an identifier
 - Cannot use reserved words for identifiers
- Reserved words (keywords)
 - Special words: part of the JavaScript language syntax

Assigning Variable Names (cont'd.)

abstract	do	if	private	true
boolean	double	implements	protected	try
break	else	import	public	typeof
byte	enum	in	return	var
case	export	instanceof	short	void
catch	extends	int	static	volatile
char	false	interface	super	while
class	final	let	switch	with
const	finally	long	synchronized	yield
continue	float	native	this	
debugger	for	new	throw	
default	function	null	throws	
delete	goto	package	transient	

Figure 1-10 JavaScript reserved words

Assigning Variable Names (cont'd.)

- Declaring and initializing variables
 - Use reserved keyword `var` to create variables
 - Example: `var curTime;`
 - Initialize variable using the following syntax:
 - `var variable_name = value;`
 - Assignment operator: equal sign (`=`)
 - Assigns value on the right side of the expression to the variable on the left side of the expression
 - Assigning a literal string value to a variable
 - Enclose text in quotation marks
 - Can assign the value of one variable to another

Assigning Variable Names (cont'd.)

- Displaying variables: print a variable
 - Pass variable name to `document.write()` method
 - Do not enclose it in quotation marks

Code: `document.write("<p>Your sales total is $" + salesTotal +
".</p>");`

Result in browser: Your sales total is \$47.58.

Assigning Variable Names (cont'd.)

- Displaying variables (cont'd.)
 - Use a plus sign to perform arithmetic operations involving variables containing numeric values

Code:

```
1  var salesTotal = 47.58;  
2  var shipping = 10;  
3  var grandTotal = salesTotal + shipping;  
4  document.write("<p>Your sales total plus shipping is $" + ↵  
5    grandTotal + ".</p>");
```

**Result in
browser:**

Your sales total plus shipping is \$57.58.

Assigning Variable Names (cont'd.)

- Modifying variables
 - Change a variable's value at any point in a script
 - Use a statement including the variable's name
 - Followed by an equal sign
 - Followed by the value to assign to the variable

Assigning Variable Names (cont'd.)

Code:

```
1  var salesTotal = 47.58;  
2  document.write("<p>Your sales total is $" + ↵  
3    salesTotal + ".</p>");  
4  var shipping = 10;  
5  salesTotal = salesTotal + shipping;  
6  document.write("<p>Your sales total plus shipping is $" + ↵  
7    salesTotal + ".</p>");
```

**Result in
browser:**

Your sales total is \$47.58.
Your sales total plus shipping is \$57.58.

Building Expressions

- Expression
 - Literal value or variable or a combination of literal values, variables, operators, and other expressions
 - Evaluated by JavaScript interpreter to produce a result
- Operands
 - Variables and literals contained in an expression
- Literal
 - Value such as a literal string or a number
- Operators
 - Symbols used in expressions to manipulate operands

Understanding Events

- Event
 - Specific circumstance monitored by JavaScript
 - Script can respond to in some way
 - Allows users to interact with web pages
- Common events: actions users perform
- Can also monitor events not resulting from user actions

Understanding Events (cont'd.)

EVENT	KEYBOARD TRIGGER	MOUSE TRIGGER	TOUCHSCREEN TRIGGER
blur	An element, such as a radio button, becomes inactive		
change	The value of an element, such as a text box, changes		
click	A user presses a key when an element is selected	A user clicks an element once	A user touches an element and then stops touching it
error	An error occurs when a document or image is being loaded		
focus	An element, such as a command button, becomes active		
keydown	A user presses a key		
keyup	A user releases a key		
load	A document or image loads		
mouseout		A user moves the mouse pointer off an element	A user stops touching an element
mouseover		A user moves the mouse pointer over an element	A user touches an element
reset	A form's fields are reset to its default values		
select	A user selects text		
submit	A user submits a form		
touchend			A user removes finger or stylus from the screen
touchmove			A finger or stylus already touching the screen moves on the screen
touchstart			A user touches a finger or stylus to the screen
unload	A document unloads		

Table 1-2 JavaScript events

Understanding Events (cont'd.)

- Working with elements and events
 - Events: associated with HTML elements
 - Event handler
 - Code that executes in response to a specific event on a specific element
 - JavaScript code for an event handler
 - Can be specified as attribute of element that initiates event

Understanding Events (cont'd.)

ELEMENT	EVENT-RELATED ATTRIBUTES
a	onfocus, onblur, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, ontouchstart, ontouchend
img	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, ontouchstart, ontouchmove, ontouchend
body	onload, onunload, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
form	onsubmit, onreset, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
input	tabindex, accesskey, onfocus, onblur, onselect, onchange, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, ontouchstart, ontouchmove, ontouchend
textarea	onfocus, onblur, onselect, onchange, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, ontouchstart, ontouchmove, ontouchend
select	onfocus, onblur, onchange, ontouchstart, ontouchend

Table 1-3 HTML elements and some of their associated events

Understanding Events (cont'd.)

- Referencing web page elements
 - Use the `getElementById()` method
 - Method of the `Document` object
 - Uses element's `id` value
 - Specific element properties
 - Appended to the element reference
 - Allows for the retrieval of information about an element or the ability to change the values assigned to its attributes

Structuring JavaScript Code

- Adding JavaScript code to a document
 - Must follow certain rules regarding placement and organization of that code

Including a script Element for Each Code Section

- Can include as many script sections as desired
 - Must include a `script` element for each section
 - Example code below
 - See Figure 1-13 for results

```
1  <h2>Sales Total</h2>
2  <script>
3      var salesTotal = 47.58;
4      document.write("<p>Your sales total is $" + salesTotal +<br>
5          "</p>");
6  </script>
7  <h2>Sales Total with Shipping</h2>
8  <script>
9      var shipping = 10;
10     salesTotal = salesTotal + shipping;
11     document.write("<p>Your sales total plus shipping is $" +<br>
12         salesTotal + "</p>");
13  </script>
```

Placing JavaScript in the Document Head or Document Body

- script element placement varies
 - Can place in the document head or document body
 - Usually placed at end of body section before</body>
 - Statements rendered in the order in which they appear in the document
 - Statements in head prevent rest of page from rendering

Creating a JavaScript Source File

- External file containing JavaScript code
 - Usually designated by the `.js` file extension
 - Can technically have any extension
 - Contains only JavaScript statements
 - No `script` element and no HTML elements
 - Use the `src` attribute of the `script` element
- Advantages
 - Neater code; code sharing; ability to hide JavaScript code from incompatible browsers
- Can use embedded JavaScript code and JavaScript source files combination

Working with Libraries

- Libraries: especially useful generic scripts used on different websites
 - Often developed by single programmer or team
 - Many available for free reuse
- Common libraries
 - Node.js
 - Backbone.js
 - Modernizr

Validating Web Pages

- Validating parser
 - Checks for a well formed web page
 - Verifies document conforms to a specific DTD
- Validation
 - Process of verifying a well-formed document and checking the elements in your document
- Web development tools offer validation capabilities
- Validating services found online
 - W3C Markup Validation Service:
 - <http://validator.w3.org>

Writing Valid XHTML Code with JavaScript

- JavaScript statements contain symbols
 - Prevents XHTML document from being well formed
- HTML handles successfully
 - `script` element statements interpreted as character data
 - Character data (CDATA)
 - Section of a document not interpreted as markup
- XHTML documents
 - `script` element statements interpreted as markup
 - Parsed character data (PCDATA)

Writing Valid JavaScript Code (cont'd.)

- JavaScript code in an XHTML document
 - Treated as PCDATA
 - Validation fails
- Two options to resolve validation issue
 - Move code into a source file
 - Keep JavaScript code within the document
 - Enclose code within a `<script>` element within a CDATA

```
<! [CDATA[  
    statements to mark as CDATA  
]]>
```

Summary

- Hypertext linking: allows quick opening of web pages
- HTML5 is current version
- Web built on a two-tier client/server system
- JavaScript programming language allows for interactive web pages and sites
 - script element tells web browser to interpret the commands it contains
 - Can save JavaScript code in a source file
- Validating parser verifies a web page

Javascript

Chapter 2
*Working with Functions, Data Types,
and Operators*

Objectives

When you complete this chapter, you will be able to:

- Use functions to organize your JavaScript code
- Use expressions and operators
- Identify the order of operator precedence in an expression

Working with Functions

- Methods
 - Procedures associated with an object
- Functions
 - Related group of JavaScript statements
 - Executed as a single unit
 - Virtually identical to methods
 - Not associated with an object
 - Must be contained within a `script` element

Defining Functions

- Named function
 - Related statements assigned a name
 - Call, or reference, named function to execute it
- Anonymous function
 - Related statements with no name assigned
 - Work only where they are located in code
- Use named function when you want to reuse code
- Use anonymous function for code that runs only once

Defining Functions (cont'd.)

- Function definition
 - Lines making up a function
- Named function syntax

```
function name_of_function(parameters) {  
    statements;  
}
```

- Anonymous function syntax

```
function (parameters) {  
    statements;  
}
```

Defining Functions (cont'd.)

- Parameter
 - Variable used within a function
 - Placed within parentheses following a function name
 - Multiple parameters allowed

`calculateVolume(length, width, height)`

Defining Functions (cont'd.)

- Function statements
 - Do the actual work
 - Contained within function braces
- Put functions in an external .js file
 - Reference at bottom of body section

```
function calculateVolume(length, width, height) {  
    var volume = length * width * height;  
    document.write(volume);  
}
```

Calling Functions

- To execute a named function:
 - Must invoke, or call, it
- Function call
 - Code calling a function
 - Consists of function name followed by parentheses
 - Contains any variables or values assigned to the function parameters
- Arguments (actual parameters)
 - Variables (values) placed in the function call statement parentheses

Calling Functions (cont'd.)

- Passing arguments
 - Sending arguments to parameters of a called function
 - Argument value assigned to the corresponding parameter value in the function definition

Calling Functions (cont'd.)

- Handling events

- Three options

- Specify function as value for HTML attribute

```
<input type="submit" onclick="showMessage()" />
```

- Specify function as property value for object

```
document.getElementById("submitButton").onclick =←  
    showMessage;
```

- Use addEventListener() method

```
var submit = document.getElementById("submitButton");  
  
submit.addEventListener("click", showMessage, false);
```

Calling Functions (cont'd.)

- Adding an event listener is most flexible
 - Separates HTML and JavaScript code
 - Can specify several event handlers for a single event
- IE8 requires use of the `attachEvent()` method instead of `addEventListener()` (see Chapter 3)

Locating Errors with the Browser Console

- Unintentional coding mistakes keep code from working
 - Browsers generate error messages in response
 - Messages displayed in browser console pane
 - Hidden by default to avoid alarming users
- Developers display browser console to see errors

BROWSER	KEYBOARD SHORTCUT	MENU STEPS
Internet Explorer	F12 , then Ctrl + 2	Click the Tools button, click F12 Developer Tools on the menu, and then in the window that opens, click the Console button.
Firefox	Ctrl + Shift + K (Win) option + command + K (Mac)	Click the Firefox button (Win) or Tools (Mac or Win), point to Web Developer , and then click Web Console .
Chrome	Ctrl + Shift + J (Win) option + command + J (Mac)	Click the Customize and control Google Chrome button, point to Tools , and then click JavaScript console .

Locating Errors with the Browser Console (cont'd.)

- Consoles specify a line number with each error



Figure 2-3: Internet Explorer browser console



Figure 2-4: Chrome browser console

Using Return Statements

- Can return function value to a calling statement
- Return statement
 - Returns a value to the statement calling the function
 - Use the `return` keyword with the variable or value to send to the calling statement
- Example:

```
function averageNumbers(a, b, c) {  
    var sum_of_numbers = a + b + c;  
    var result = sum_of_numbers / 3;  
    return result;  
}
```

Understanding Variable Scope

- **Variable scope**
 - Where in code a declared variable can be used
- **Global variable**
 - Declared outside a function
 - Available to all parts of code
- **Local variable**
 - Declared inside a function
 - Only available within the function in which it is declared
 - Cease to exist when the function ends
 - Keyword `var` required

Understanding Variable Scope (cont'd.)

- Good programming technique
 - Always use the `var` keyword when declaring variables
 - Clarifies where and when variable used
- Poor programming technique
 - Declaring a global variable inside of a function by not using the `var` keyword
 - Harder to identify global variables in your scripts

Understanding Variable Scope (cont'd.)

- If variable declared within a function and does not include the `var` keyword
 - Variable automatically becomes a global variable
- Program may contain global and local variables with the same name
 - Local variable takes precedence
 - Value assigned to local variable of the same name
 - Not assigned to global variable of the same name

Understanding Variable Scope (cont'd.)

```
var color = "green";  
  
function duplicateVariableNames() {  
  
    var color = "purple";  
  
    document.write(color);  
  
    // value printed is purple  
}  
  
duplicateVariableNames();  
  
document.write(color);  
  
// value printed is green
```

Using Built-in JavaScript Functions

- Called the same way a custom function is called

FUNCTION	DESCRIPTION
<code>decodeURI(string)</code>	Decodes text strings encoded with <code>encodeURI()</code>
<code>decodeURIComponent(string)</code>	Decodes text strings encoded with <code>encodeURIComponent()</code>
<code>encodeURI(string)</code>	Encodes a text string so it becomes a valid URI
<code>encodeURIComponent(string)</code>	Encodes a text string so it becomes a valid URI component
<code>eval(string)</code>	Evaluates expressions contained within strings
<code>isFinite(number)</code>	Determines whether a number is finite
<code>isNaN(number)</code>	Determines whether a value is the special value NaN (Not a Number)
<code>parseFloat(string)</code>	Converts string literals to floating-point numbers
<code>parseInt(string)</code>	Converts string literals to integers

Table 2-2 Built-in JavaScript functions

Working with Data Types

- Data type
 - Specific information category a variable contains
- Primitive types
 - Data types assigned a single value

DATA TYPE	DESCRIPTION
number	A positive or negative number with or without decimal places, or a number written using exponential notation
Boolean	A logical value of <code>true</code> or <code>false</code>
string	Text such as "Hello World"
<code>undefined</code>	An unassigned, undeclared, or nonexistent value
<code>null</code>	An empty value

Table 2-3 Primitive JavaScript data types

Working with Data Types (cont'd.)

- The `null` value: data type and a value
 - Can be assigned to a variable
 - Indicates no usable value
 - Use: ensure a variable does not contain any data
- Undefined variable
 - Never had a value assigned to it, has not been declared, or does not exist
 - Indicates variable never assigned a value: not even `null`
 - Use: determine if a value being used by another part of a script

Working with Data Types (cont'd.)

```
var stateTax;  
  
document.write(stateTax);  
  
stateTax = 40;  
  
document.write(stateTax);  
  
stateTax = null;  
  
document.write(stateTax);
```

undefined
40
null

Figure 2-7 Variable assigned values of undefined and null

Working with Data Types (cont'd.)

- Strongly typed programming languages
 - Require declaration of the data types of variables
 - Strong typing also known as static typing
 - Data types do not change after declared
- Loosely typed programming languages
 - Do not require declaration of the data types of variables
 - Loose typing also known as dynamic typing
 - Data types can change after declared

Working with Data Types (cont'd.)

- JavaScript interpreter automatically determines data type stored in a variable
- Examples:

```
diffTypes = "Hello World"; // String  
diffTypes = 8;           // Integer number  
diffTypes = 5.367;       // Floating-point number  
diffTypes = true;        // Boolean  
diffTypes = null;        // Null
```

Understanding Numeric Data Types

- JavaScript supports two numeric data types
 - Integers and floating-point numbers
- Integer
 - Positive or negative number with no decimal places
- Floating-point number
 - Number containing decimal places or written in exponential notation
 - Exponential notation (scientific notation)
 - Shortened format for writing very large numbers or numbers with many decimal places

Using Boolean Values

- Logical value of true or false
 - Used for decision making
 - Which parts of a program should execute
 - Used for comparing data
- JavaScript programming Boolean values
 - The words `true` and `false`
 - JavaScript converts `true` and `false` values to the integers 1 and 0 when necessary

Using Boolean Values (cont'd.)

```
1 var newCustomer = true;  
2 var contractorRates = false;  
3 document.write("<p>New customer: " + newCustomer + "</p>");  
4 document.write("<p>Contractor rates: " + contractorRates + "  
5 "</p>");
```



New customer: true
Contractor rates: false

Figure 2-9 Boolean values

Working with Strings

- Text string
 - Contains zero or more characters
 - Surrounded by double or single quotation marks
 - Can be used as literal values or assigned to a variable
- Empty string
 - Zero-length string value
 - Valid for literal strings
 - Not considered to be `null` or `undefined`

Working with Strings (cont'd.)

- To include a quoted string within a literal string surrounded by double quotation marks
 - Surround the quoted string with single quotation marks
- To include a quoted string within a literal string surrounded by single quotation marks
 - Surround the quoted string with double quotation marks
- String must begin and end with the same type of quotation marks

Working with Strings (cont'd.)

```
document.write("<h1>Speech at the Berlin Wall<br>
(excerpt)</h1>");

document.write("<p>Two thousand years ago, the proudest boast<br>
was 'civis Romanus sum.'<br />");

document.write('Today, in the world of freedom, the proudest<br>
boast is "Ich bin ein Berliner."</p>');

var speaker = "<p>John F. Kennedy</p>";

var date = 'June 26, 1963';

document.write(speaker);
document.write(date);
```



Figure 2-10 String examples in a browser

Working with Strings (cont'd.)

- String operators
 - Concatenation operator (+): combines two strings

```
var destination = "Honolulu";  
  
var location = "Hawaii";  
  
destination = destination + " is in " + location;
```

- Compound assignment operator (+=): combines two strings

```
var destination = "Honolulu";  
  
destination += " is in Hawaii";
```

- Plus sign
 - Concatenation operator and addition operator

Working with Strings (cont'd.)

- Escape characters and sequences
 - Escape character
 - Tells the compiler or interpreter that the character that follows has a special purpose
 - In JavaScript, escape character is backslash (\)
 - Escape sequence
 - Escape character combined with other characters
 - Most escape sequences carry out special functions

Working with Strings (cont'd.)

ESCAPE SEQUENCE	CHARACTER
\\	Backslash
\b	Backspace
\r	Carriage return
\"	Double quotation mark
\f	Form feed
\t	Horizontal tab
\n	Newline
\0	Null character
\'	Single quotation mark (apostrophe)
\v	Vertical tab
\xXX	Latin-1 character specified by the XX characters, which represent two hexadecimal digits
\uXXXX	Unicode character specified by the XXXX characters, which represent four hexadecimal digits

Table 2-4 JavaScript escape sequences

Using Operators to Build Expressions

OPERATOR TYPE	OPERATORS	DESCRIPTION
Arithmetic	addition (+) subtraction (-) multiplication (*) division (/) modulus (%) increment (++) decrement (--) negation (-)	Perform mathematical calculations
Assignment	assignment (=) compound addition assignment (+=) compound subtraction assignment (-=) compound multiplication assignment (*=) compound division assignment (/=) compound modulus assignment (%=)	Assign values to variables
Comparison	equal (==) strict equal (===) not equal (!=) strict not equal (!==) greater than (>) less than (<) greater than or equal (>=) less than or equal (<=)	Compare operands and return a Boolean value

Table 2-5 JavaScript operator types (*continues*)

Using Operators to Build Expressions (cont'd.)

OPERATOR TYPE	OPERATORS	DESCRIPTION
Logical	And (<code>&&</code>) Or (<code> </code>) Not (<code>!</code>)	Perform Boolean operations on Boolean operands
String	concatenation (<code>+</code>) compound concatenation assignment (<code>+=</code>)	Perform operations on strings
Special	property access (<code>.</code>) array index (<code>[]</code>) function call (<code>()</code>) comma (<code>,</code>) conditional expression (<code>? :</code>) <code>delete</code> (<code>delete</code>) property exists (<code>in</code>) object type (<code>instanceof</code>) new object (<code>new</code>) data type (<code>typeof</code>) <code>void</code> (<code>void</code>)	Various purposes; do not fit within other operator categories

Table 2-5 JavaScript operator types (cont'd.)

Using Operators to Build Expressions (cont'd.)

- Binary operator
 - Requires an operand before and after the operator
- Unary operator
 - Requires a single operand either before or after the operator

Arithmetic Operators

- Perform mathematical calculations
 - Addition, subtraction, multiplication, division
 - Returns the modulus of a calculation
- Arithmetic binary operators

NAME	OPERATOR	DESCRIPTION
Addition	+	Adds two operands
Subtraction	-	Subtracts one operand from another operand
Multiplication	*	Multiplies one operand by another operand
Division	/	Divides one operand by another operand
Modulus	%	Divides one operand by another operand and returns the remainder

Table 2-6 Arithmetic binary operators

Arithmetic Operators (cont'd.)

- Arithmetic binary operators (cont'd.)
 - Value of operation on right side of the assignment operator assigned to variable on the left side
 - Example: `arithmeticValue = x + y;`
 - Result assigned to the `arithmeticValue` variable
 - Division operator (/)
 - Standard mathematical division operation
 - Modulus operator (%)
 - Returns the remainder resulting from the division of two integers

Arithmetic Operators (cont'd.)

```
var divisionResult = 15 / 6;  
  
var modulusResult = 15 % 6;  
  
document.write("<p>15 divided by 6 is "<br>  
+ divisionResult + ".</p>"); // prints '2.5'  
  
document.write("<p>The whole number 6 goes into 15 twice,<br>  
with a remainder of " + modulusResult + ".</p>"); // prints '3'
```

15 divided by 6 is 2.5.

The whole number 6 goes into 15 twice, with a remainder of 3.

Figure 2-13 Division and modulus expressions

Arithmetic Operators (cont'd.)

- Arithmetic binary operators (cont'd.)
 - Assignment statement
 - Can include combination of variables and literal values on the right side
 - Cannot include a literal value as the left operand
 - JavaScript interpreter
 - Attempts to convert the string values to numbers
 - Does not convert strings to numbers when using the addition operator

Arithmetic Operators (cont'd.)

- Prefix operator
 - Placed before a variable
- Postfix operator
 - Placed after a variable

NAME	OPERATOR	DESCRIPTION
Increment	<code>++</code>	Increases an operand by a value of one
Decrement	<code>--</code>	Decreases an operand by a value of one
Negation	<code>-</code>	Returns the opposite value (negative or positive) of an operand

Table 2-7 Arithmetic unary operators

Arithmetic Operators (cont'd.)

- Arithmetic unary operators
 - Performed on a single variable using unary operators
 - Increment (++) unary operator: used as prefix operators
 - Prefix operator placed before a variable
 - Decrement (--) unary operator: used as postfix operator
 - Postfix operator placed after a variable
 - Example: `++count;` and `count++;`
 - Both increase the count variable by one, but return different values

Arithmetic Operators (cont'd.)

```
1  var studentID = 100;  
2  var curStudentID;  
3  curStudentID = ++studentID; // assigns '101'  
4  document.write("<p>The first student ID is " +  
5      curStudentID + "</p>");  
6  curStudentID = ++studentID; // assigns '102'  
7  document.write("<p>The second student ID is " +  
8      curStudentID + "</p>");  
9  curStudentID = ++studentID; // assigns '103'  
10 document.write("<p>The third student ID is " +  
11      curStudentID + "</p>");
```

The first student ID is 101

The second student ID is 102

The third student ID is 103

Figure 2-14 Output of the prefix version of the student ID script

Arithmetic Operators (cont'd.)

```
1  var studentID = 100;  
2  var curStudentID;  
3  curStudentID = studentID++; // assigns '100'  
4  document.write("<p>The first student ID is " + ↵  
5    curStudentID + "</p>");  
6  curStudentID = studentID++; // assigns '101'  
7  document.write("<p>The second student ID is " + ↵  
8    curStudentID + "</p>");  
9  curStudentID = studentID++; // assigns '102'  
10 document.write("<p>The third student ID is " + ↵  
11   curStudentID + "</p>");
```

The first student ID is 100

The second student ID is 101

The third student ID is 102

Figure 2-15 Output of the postfix version of the student ID script

Assignment Operators

- Used for assigning a value to a variable
- Equal sign (=)
 - Assigns initial value to a new variable
 - Assigns new value to an existing variable
- Compound assignment operators
 - Perform mathematical calculations on variables and literal values in an expression
 - Then assign a new value to the left operand

Assignment Operators (cont'd.)

NAME	OPERATOR	DESCRIPTION
Assignment	=	Assigns the value of the right operand to the left operand
Compound addition assignment	+=	Combines the value of the right operand with the value of the left operand (if the operands are strings), or adds the value of the right operand to the value of the left operand (if the operands are numbers), and assigns the new value to the left operand
Compound subtraction assignment	-=	Subtracts the value of the right operand from the value of the left operand, and assigns the new value to the left operand
Compound multiplication assignment	*=	Multiplies the value of the right operand by the value of the left operand, and assigns the new value to the left operand
Compound division assignment	/=	Divides the value of the left operand by the value of the right operand, and assigns the new value to the left operand
Compound modulus assignment	%=	Divides the value of the left operand by the value of the right operand, and assigns the remainder (the modulus) to the left operand

Table 2-8 Assignment operators

Assignment Operators (cont'd.)

- `+=` compound addition assignment operator
 - Used to combine two strings and to add numbers
- Examples:

```
1  var x, y;
2  // += operator with string values
3  x = "Hello ";
4  x += "World"; // x changes to "Hello World"
5  // += operator with numeric values
6  x = 100;
7  y = 200;
8  x += y; // x changes to 300
9  // -= operator
10 x = 10;
11 y = 7;
12 x -= y; // x changes to 3
13 // *= operator
14 x = 2;
15 y = 6;
16 x *= y; // x changes to 12
```

Assignment Operators (cont'd.)

- Examples: (cont'd.)

```
17 // /= operator
18 x = 24;
19 y = 3;
20 x /= y; // x changes to 8
21 // %= operator
22 x = 3;
23 y = 2;
24 x %= y; // x changes to 1
25 // *= operator with a number and a convertible string
26 x = "100";
27 y = 5;
28 x *= y; // x changes to 500
29 // *= operator with a number and a nonconvertible string
30 x = "one hundred";
31 y = 5;
32 x *= y; // x changes to NaN
```

Comparison and Conditional Operators

- Comparison operators
 - Compare two operands
 - Determine if one numeric value is greater than another
 - Boolean value of true or false returned after compare
- Operands of comparison operators
 - Two numeric values: compared numerically
 - Two nonnumeric values: compared in alphabetical order
 - Number and a string: convert string value to a number
 - If conversion fails: value of false returned

Comparison and Conditional Operators (cont'd.)

NAME	OPERATOR	DESCRIPTION
Equal	==	Returns true if the operands are equal
Strict equal	===	Returns true if the operands are equal and of the same type
Not equal	!=	Returns true if the operands are not equal
Strict not equal	!==	Returns true if the operands are not equal or not of the same type
Greater than	>	Returns true if the left operand is greater than the right operand
Less than	<	Returns true if the left operand is less than the right operand
Greater than or equal	>=	Returns true if the left operand is greater than or equal to the right operand
Less than or equal	<=	Returns true if the left operand is less than or equal to the right operand

Table 2-9 Comparison operators

Comparison and Conditional Operators (cont'd.)

- Conditional operator
 - Executes one of two expressions based on conditional expression results
 - Syntax

conditional expression ? expression1 : expression2;

- If conditional expression evaluates to true:
 - Then expression1 executes
- If the conditional expression evaluates to false:
 - Then expression2 executes

Comparison and Conditional Operators (cont'd.)

- Example of conditional operator:

```
var intVariable = 150;  
  
var result;  
  
intVariable > 100 ?←  
    result = "intVariable is greater than 100" :←  
    result = "intVariable is less than or equal to 100";  
  
document.write(result);
```

Falsy and Truthy Values

- Six falsy values treated like Boolean `false`:
 - `""`
 - `-0`
 - `0`
 - `NaN`
 - `null`
 - `undefined`
- All other values are truthy, treated like Boolean `true`

Logical Operators

- Compare two Boolean operands for equality

NAME	OPERATOR	DESCRIPTION
And	&&	Returns <code>true</code> if both the left operand and right operand return a value of <code>true</code> ; otherwise, it returns a value of <code>false</code>
Or		Returns <code>true</code> if either the left operand or right operand returns a value of <code>true</code> ; if neither operand returns a value of <code>true</code> , then the expression containing the <code>Or </code> operator returns a value of <code>false</code>
Not	!	Returns <code>true</code> if an expression is <code>false</code> , and returns <code>false</code> if an expression is <code>true</code>

Table 2-10 Logical operators

Special Operators

NAME	OPERATOR	DESCRIPTION
Property access	.	Appends an object, method, or property to another object
Array index	[]	Accesses an element of an array
Function call	()	Calls up functions or changes the order in which individual operations in an expression are evaluated
Comma	,	Allows you to include multiple expressions in the same statement
Conditional expression	? :	Executes one of two expressions based on the results of a conditional expression
Delete	<code>delete</code>	Deletes array elements, variables created without the <code>var</code> keyword, and properties of custom objects
Property exists	<code>in</code>	Returns a value of <code>true</code> if a specified property is contained within an object
Object type	<code>instanceof</code>	Returns <code>true</code> if an object is of a specified object type
New object	<code>new</code>	Creates a new instance of a user-defined object type or a predefined JavaScript object type
Data type	<code>typeof</code>	Determines the data type of a variable
Void	<code>void</code>	Evaluates an expression without returning a result

Table 2-11 Special operators

Special Operators (cont'd.)

RETURN VALUE	RETURNED FOR
number	Integers and floating-point numbers
string	Text strings
boolean	True or false
object	Objects, arrays, and null variables
function	Functions
undefined	Undefined variables

Table 2-12 Values returned by `typeof` operator

Understanding Operator Precedence

- Operator precedence
 - Order in which operations in an expression evaluate
- Associativity
 - Order in which operators of equal precedence execute
 - Left to right associativity
 - Right to left associativity

Understanding Operator Precedence (cont'd.)

- Evaluating associativity
 - Example: multiplication and division operators
 - Associativity of left to right

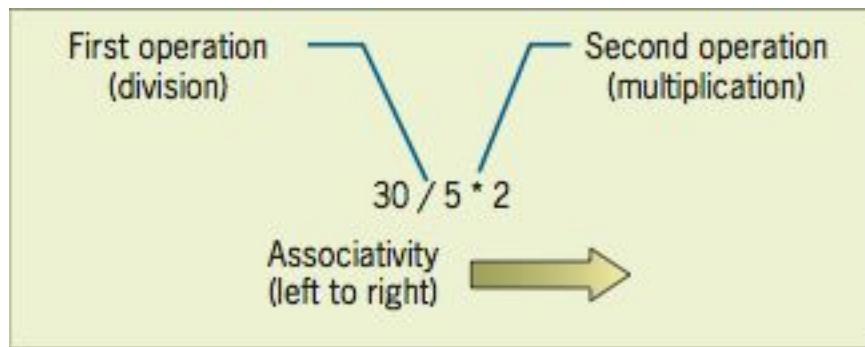


Figure 2-16 Conceptual illustration
of left to right associativity

Understanding Operator Precedence (cont'd.)

- Evaluating associativity (cont'd.)
 - Example: Assignment operator and compound assignment operators
 - Associativity of right to left
 - $x = y *= ++x$

```
var x = 3;  
var y = 2;  
x = y *= ++x;
```

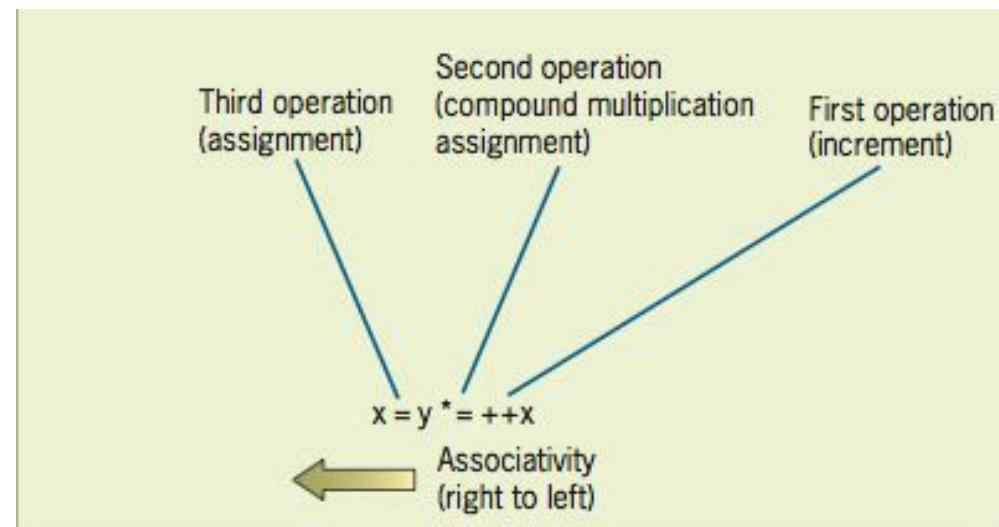


Figure 2-17 Conceptual illustration of right-to-left associativity

Summary

- Functions
 - Similar to methods associated with an object
 - Pass parameters
 - To execute, must be called
- Variable scope
 - Where a declared variable can be used
 - Global and local variables
- Data type
 - Specific category of information a variable contains
 - Static typing and dynamic typing

Summary (cont'd.)

- Numeric data types: integer and floating point
- Boolean values: true and false
- Strings: one or more character surrounded by double or single quotes
 - String operators
 - Escape character
- Operators build expressions
- Operator precedence
 - Order in which operations in an expression are evaluated

Javascript

Chapter 3
Building Arrays and Controlling Flow

Objectives

In this chapter, you will:

- Store data in arrays
- Use `while` statements, `do/while` statements, and `for` statements to repeatedly execute code
- Use `continue` statements to restart looping statements
- Use `if` statements, `if/else` statements, and `switch` statements to make decisions
- Nest one `if` statement in another

Storing Data in Arrays

- Array
 - Set of data represented by a single variable name



Figure 3-1 Conceptual illustration of an array

Declaring and Initializing Arrays

- Array literal
 - most common way to create an array
 - declares a variable and specifies array as content
- Syntax

```
var name = [value1, value2, value3, ...];
```

- Example:
 - Create an array named newsSections containing 4 strings as elements

```
var newsSections = ["world", "local", "opinion", "sports"]
```

Declaring and Initializing Arrays (cont'd.)

- Element
 - Each piece of data contained in an array
- Index
 - Element's numeric position within the array
 - Array element numbering
 - Starts with index number of zero (0)
- Reference element using its index number
 - Example: to reference the 2nd element in the newsSections array

`newsSections [1]`

Declaring and Initializing Arrays (cont'd.)

- Assigning values to individual array elements
 - Include the array index for an individual element
- Example:
 - Add value "entertainment" as fifth element of newsSections array

```
newsSections [4] = "entertainment";
```

Declaring and Initializing Arrays (cont'd.)

- Can create an array without any elements
 - Add new elements as necessary
 - Array size can change dynamically

```
var colors = [] ;  
  
colors[2] = "yellow" ;
```

- JavaScript values assigned to array elements
 - Can be different data types

Accessing Element Information

- To access an element's value:
 - Include brackets and element index
- Examples:

```
var sec1Head = document.getElementById("section1");
```

```
var sec2Head = document.getElementById("section2");
```

```
var sec3Head = document.getElementById("section3");
```

```
sec1Head.innerHTML = newsSections[0]; // "world"
```

```
sec2Head.innerHTML = newsSections[1]; // "local"
```

```
sec3Head.innerHTML = newsSections[2]; // "opinion"
```

Modifying Elements

- To modify values in existing array elements
 - Include brackets and element index
- Can change a value assigned to an array element
- Example:

```
newsSections[4] = "living";
```

Determining the Number of Elements in an Array

- `length` property
 - Returns the number of elements in an array
- Syntax

name.length;

Using the Array Object

- JavaScript represents arrays with the `Array` object
 - Contains a special constructor named `Array()`
- Constructor
 - Special function type used as the basis for creating reference variables
- Syntax

```
var newsSections = new Array(6);
```

- Array literals preferred
 - Easier

Referencing Default Collections of Elements

- `getElementsByName()` method
 - Can reference web page element by looking up all elements of a certain type in document and referencing one element in that collection
 - Resulting collection uses syntax similar to arrays
- Example:

```
document.getElementsByName("li") [2]
```

Repeating Code

- Loop statement
 - Control flow statement repeatedly executing a statement or a series of statements
 - While a specific condition is true or until a specific condition becomes true
- Three types of loop statements
 - `while` statements
 - `do/while` statements
 - `for` statements

while Statements

- **while statement**
 - Repeats a statement or series of statements
 - As long as a given conditional expression evaluates to a truthy value

- **Syntax**

```
while (expression) {  
    statements  
}
```

- **Iteration**
 - Each repetition of a looping statement

while Statements (cont'd.)

- Counter
 - Variable incremented or decremented with each loop statement iteration
- Examples:
 - while statement using an increment operator
 - while statement using a decrement operator
 - while statement using the *= assignment operator

while Statements (cont'd.)

```
var count = 1;  
while (count <= 5) {  
    document.write(count + "<br />");  
    count++;  
}  
document.write("<p>You have printed 5 numbers.</p>");
```

Result in browser:

1
2
3
4
5

You have printed 5 numbers.

while Statements (cont'd.)

```
var count = 10;  
while (count > 0) {  
    document.write(count + "<br />");  
    count--;  
}  
document.write("<p>We have liftoff.</p>");
```

Result in browser:



```
10  
9  
8  
7  
6  
5  
4  
3  
2  
1  
We have liftoff.
```

while Statements (cont'd.)

```
var count = 1;  
while (count <= 100) {  
    document.write(count + "<br />");  
    count *= 2;  
}
```

Result in browser:

```
1  
2  
4  
8  
16  
32  
64
```

while Statements (cont'd.)

- Infinite loop
 - Loop statement that never ends
 - Conditional expression: never false
 - Example:

```
var count = 1;  
while (count <= 10) {  
    window.alert("The number is " + count + ".");  
}
```

while Statements (cont'd.)

- Example:
 - assigning array element values to table cells:

```
function addColumnHeaders() {  
    var i = 0;  
    while (i < 7) {  
        document.getElementsByTagName("th")←  
            [i].innerHTML = daysOfWeek[i];  
        i++;  
    }  
}
```

do/while Statements

- do/while statement
 - Executes a statement or statements once
 - Then repeats the execution as long as a given conditional expression evaluates to a truthy value
- Syntax

```
do {  
    statements;  
} while (expression);
```

do/while Statements (cont'd.)

- Examples:

```
var count = 2;  
do {  
    document.write("<p>The count is equal to " +  
        count + "</p>");  
    count++;  
} while (count < 2);
```

```
var count = 2;  
while (count < 2) {  
    document.write("<p>The count is equal to " +  
        count + "</p>");  
    count++;  
}
```

do/while Statements (cont'd.)

- Example:
 - adding days of week with a do/while statement instead of a while statement

```
var i = 0;  
do {  
    document.getElementsByTagName("th") [i].innerHTML =←  
        daysOfWeek [i];  
    i++;  
} while (i < 7);
```

for Statements

- **for statement**
 - Repeats a statement or series of statements
 - As long as a given conditional expression evaluates to a truthy value
 - Can also include code that initializes a counter and changes its value with each iteration
- **Syntax**

```
for (counter_declaration; condition;  
     counter_operation) {  
    statements  
}
```

for Statements (cont'd.)

- Steps when JavaScript interpreter encounters a `for` loop
 1. Counter variable declared and initialized
 2. `for` loop condition evaluated
 3. If condition evaluation in Step 2 returns truthy value:
 - `for` loop statements execute, Step 4 occurs, and the process starts over again with Step 2If condition evaluation in Step 2 returns falsy value:
 - `for` statement ends
 - Next statement following the `for` statement executes
 4. Update statement in the `for` statement executed

for Statements (cont'd.)

```
var brightestStars =
  ["Sirius", "Canopus", "Arcturus", "Rigel", "Vega"];
for (var count = 0; count < brightestStars.length; count++) {
  document.write(brightestStars[count] + "<br />");
}
```

Result in browser:

```
Sirius
Canopus
Arcturus
Rigel
Vega
```

for Statements (cont'd.)

- **for statement**
 - More efficient than a `while` **statement**
- **Examples:**

```
var count = 1;  
while (count < brightestStars.length) {  
    document.write(count + "<br />");  
    count++;  
}
```

```
for (var count = 1; count < brightestStars.length; count++) {  
    document.write(count + "<br />");  
}
```

for Statements (cont'd.)

- Example:
 - addColumnHeaders() function with a for statement instead of a do/while statement

```
function addColumnHeaders() {  
    for (var i = 0; i < 7; i++) {  
        document.getElementsByTagName("th") [i].innerHTML←  
            = daysOfWeek[i];  
    }  
}
```

Using `continue` Statements to Restart Execution

- `continue` statement
 - Halts a looping statement
 - Restarts the loop with a new iteration
 - Used to stop a loop for the current iteration
 - Have the loop to continue with a new iteration
- Examples:
 - `for` loop with a `continue` statement

Using continue Statements to Restart Execution (cont'd.)

```
for (var count = 1; count <= 5; count++) {  
    if (count === 3) {  
        continue;  
    }  
    document.write("<p>" + count + "</p>");  
}
```

Result in browser:

```
1  
2  
4  
5
```

Making Decisions

- Decision making
 - Process of determining the order in which statements execute in a program
- Decision-making statements, decision-making structures, or conditional statements
 - Special types of JavaScript statements used for making decisions
- `if` statement
 - Most common type of decision-making statement

if Statements

- Used to execute specific programming code
 - If conditional expression evaluation returns truthy value
- Syntax

```
if (condition) {  
    statements  
}
```

- . After the `if` statement executes:
 - Any subsequent code executes normally

`if` Statements (cont'd.)

- Use a command block to construct a decision-making structure containing multiple statements
- Command block
 - Set of statements contained within a set of braces

if/else Statements

- Executes one action if the condition is true
 - And a different action if the condition is false
- Syntax for an `if . . . else` statement

```
if (expression) {  
    statements  
}  
else {  
    statements  
}
```

if/else Statements (cont'd.)

- Example:

```
var today = "Tuesday"
if (today === "Monday") {
    document.write("<p>Today is Monday</p>") ;
}
else {
    document.write("<p>Today is not Monday</p>") ;
}
```

Nested if and if/else Statements

- Nested decision-making structures
 - One decision-making statement contains another decision-making statement
- Nested if statement
 - An if statement contained within an if statement or within an if/else statement
- Nested if/else statement
 - An if/else statement contained within an if statement or within an if/else statement

Nested if and if/else Statements (cont'd.)

- Example:

```
var salesTotal = 75;  
if (salesTotal > 50) {  
    if (salesTotal < 100) {  
        document.write("<p>The sales total is  
                    between 50 and 100.</p>");  
    }  
}
```

else if constructions

- Compact version of nested if/else statements
 - combine an else statement with its nested if statement
 - requires fewer characters
 - easier to read

else if constructions (cont'd.)

nested if/else version

```
if (gameLocation[i] === "away") {  
    paragraphs[1].innerHTML = "@ ";  
}  
else {  
    if (gameLocation[i] === "home") {  
        paragraphs[1].innerHTML = "vs ";  
    }  
}
```

else if version

```
if (gameLocation[i] === "away") {  
    paragraphs[1].innerHTML = "@ ";  
}  
else if (gameLocation[i] === "home") {  
    paragraphs[1].innerHTML = "vs ";  
}
```

else if constructions (cont'd)

- Used to create backward-compatible event listeners:

```
var submitButton = document.getElementById("button");
if (submitButton.addEventListener) {
    submitButton.addEventListener("click", submitForm, ←
        false);
}
else if (submitButton.attachEvent) {
    submitButton.attachEvent("onclick", submitForm);
}
```

switch Statements

- Controls program flow by executing a specific set of statements
 - Dependent on an expression value
- Compares expression value to value contained within a case label
- case label
 - Represents a specific value
 - Contains one or more statements that execute:
 - If case label value matches the switch statement's expression value

switch Statements (cont'd.)

- Syntax

```
switch (expression) {  
    case label:  
        statements;  
        break;  
    case label:  
        statements;  
        break;  
    ...  
    default:  
        statements;  
        break;  
}
```

switch Statements (cont'd.)

- **default label**
 - Executes when the value returned by the `switch` statement expression does not match a case label
- **When a `switch` statement executes:**
 - Value returned by the expression is compared to each case label
 - In the order in which it is encountered
- **`break` statement**
 - Ends execution of a `switch` statement
 - Should be final statement after each case label

switch Statements (cont'd.)

```
function city_location(americanCity) {  
    switch (americanCity) {  
        case "Boston":  
            return "Massachusetts";  
            break;  
        case "Chicago":  
            return "Illinois";  
            break;  
        case "Los Angeles":  
            return "California";  
            break;  
        case "Miami":  
            return "Florida";  
            break;  
        case "New York":  
            return "New York";  
            break;  
        default:  
            return "United States";  
            break;  
    }  
}  
document.write("<p>" + city_location("Boston") + "</p>");
```

Summary

- Array
 - Set of data represented by a single variable name
 - Index: element's numeric position within the array
 - Can access and modify array elements
 - `length` property
 - number of elements in an array

Summary (cont'd.)

- Loop statements
 - while statements, do/while statements, and for statements
 - Iteration: each repetition of a looping statement
 - Counter: variable
 - Incremented or decremented with each iteration of a loop statement
 - continue statement
 - Restarts a loop with a new iteration

Summary (cont'd.)

- Decision making
 - Determining the order in which statements execute in a program
- May execute in a linear fashion
 - `if statement`, `if/else statement`, `else if construction`
 - Nested decision-making structures
 - `switch statement` and `case labels`
 - `break statement`: used to exit control statements
 - Command block
 - Set of statements contained within a set of braces
 - May repeat the same statement, function, or code section

Javascript

Chapter 4
Debugging and Error Handling

Objectives

When you complete this chapter, you will be able to:

- Recognize error types
- Trace errors with dialog boxes and the console
- Use comments to locate bugs
- Trace errors with debugging tools
- Write code to respond to exceptions and errors

Introduction to Debugging

- Programming languages have syntax (rules)
- Logic
 - Order in which various program parts run (execute)
- Bug
 - Any program error
 - Causes program to function incorrectly due to incorrect syntax or flaws in logic
- Debugging
 - Process of tracing and resolving errors in a program

Understanding Syntax Errors

- Syntax errors
 - Occur when interpreter fails to recognize code
 - Causes
 - Incorrect use of JavaScript code
 - References to non-existent objects, methods, variables

Handling Run-Time Errors

- Run-time errors
 - Occur when interpreter encounters a problem while program executing
 - Not necessarily JavaScript language errors
 - Occur when interpreter encounters code it cannot execute
 - Run-time error can be caused by a syntax error

Identifying Logic Errors

- Logic errors
 - Flaw in a program's design
 - Prevents program from running as anticipated
 - “Logic” reference
 - Execution of program statements and procedures in the correct order to produce the desired results
 - Example: multiplying instead of dividing

```
var divisionResult = 10 * 2;  
document.write("Ten divided by two is equal to " +  
    divisionResult);
```

Identifying Logic Errors (cont'd.)

- Example: infinite loop

```
for (var count = 10; count >= 0; count) {  
    document.write("We have liftoff in " + count);  
}
```

- Example: infinite loop corrected

```
for (var count = 10; count >= 0; count--) {  
    document.write("We have liftoff in " + count);  
}
```

Interpreting Error Messages

- First line of defense in locating bugs
 - Browser console displays
 - Line number where error occurred
 - Error description
- Run-time errors
 - Error messages generated by a web browser
 - Can be caused by syntax errors but not by logic errors
 - Example:

```
function missingClosingBrace() {  
    var message = "This function is missing a closing brace.";  
    window.alert(message);
```

Interpreting Error Messages (cont'd.)

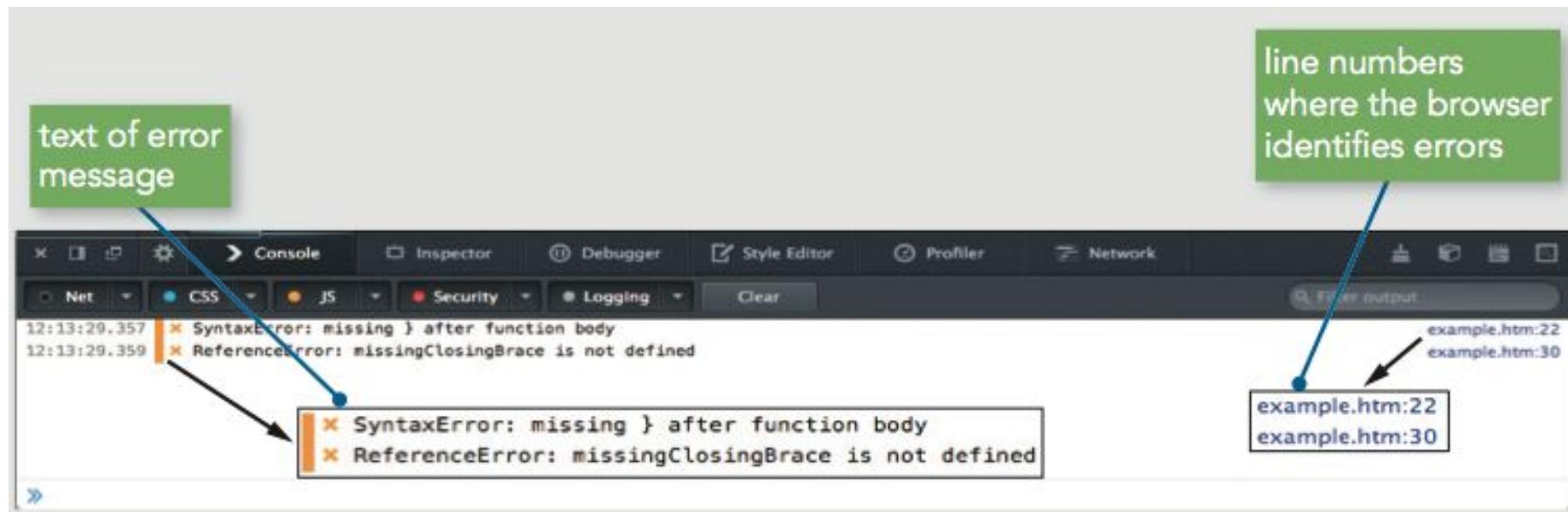


Figure 4-3 Firefox error messages

Interpreting Error Messages (cont'd.)

- Error message
 - Displays error's general location in a program
 - Not an exact indicator
- Browsers do not strictly enforce JavaScript syntax
- Mitigating bugs in JavaScript programs
 - Always use good syntax
 - Thoroughly test with every browser type, version
 - Test browser if used by more than one percent of the market

Using Basic Debugging Techniques

- Syntax errors can be difficult to pinpoint
- A few common techniques are helpful in tracking down bugs

Tracing Errors with the window.alert() Method

- Tracing
 - Examining statements in an executing program
- window.alert() method
 - A useful way to trace JavaScript code
 - Place at different points within program
 - Used to display variable or array contents or value returned from a function
- Use multiple window.alert() methods
 - Check values as code executes
- Example: function not returning correct result of 485
 - Returning 5169107

Tracing Errors with the window.alert() Method (cont'd.)

```
function calculatePay() {  
    var payRate = 15; numHours = 40;  
    var grossPay = payRate * numHours;  
    window.alert(grossPay);  
    var federalTaxes = grossPay * .06794;  
    var stateTaxes = grossPay * .0476;  
    var socialSecurity = grossPay * .062;  
    var medicare = grossPay * .0145;  
    var netPay = grossPay - federalTaxes;  
    netPay *= stateTaxes;  
    netPay *= socialSecurity;  
    netPay *= medicare;  
    return netPay;  
}
```

Tracing Errors with the window.alert() Method (cont'd.)

- Drawback
 - Must close each dialog box for code to continue executing
- Use selectively at key points
- Place debugging code at different indent level to distinguish from program code

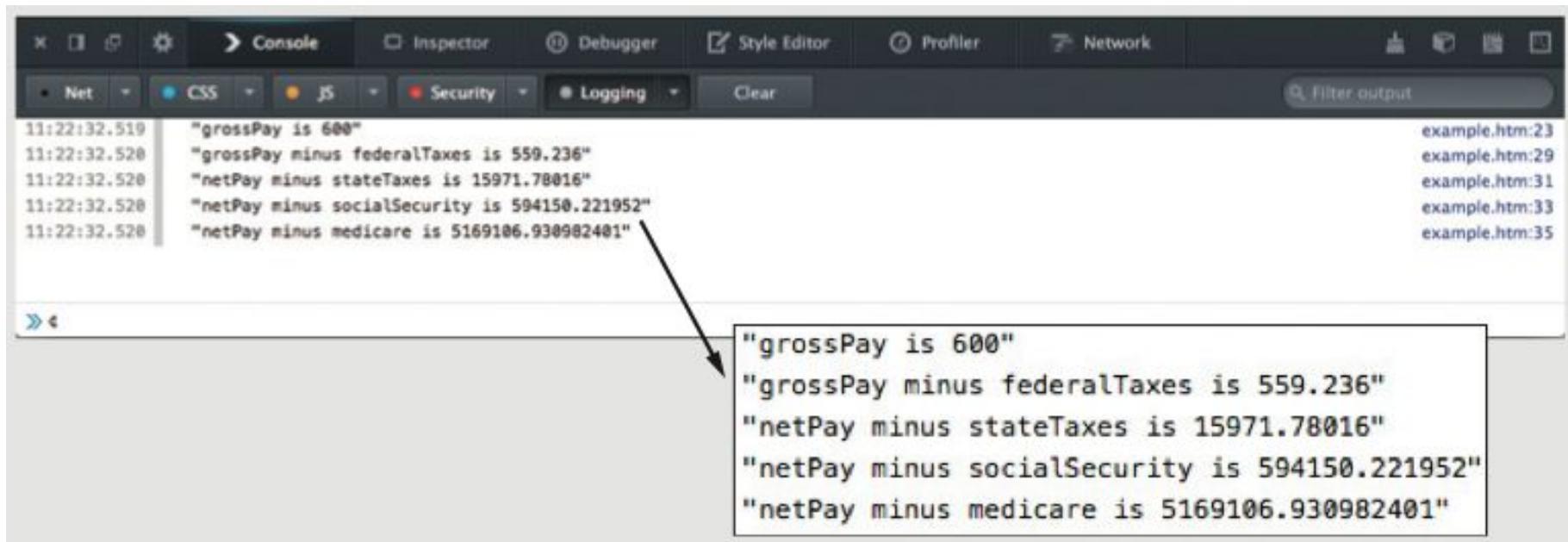
Tracing Errors with the console.log() Method

- Trace a bug by analyzing a list of values
- Logging
 - writing values directly to the console using the `console.log()` method
 - syntax: `console.log(value);`
 - can log string literal, variable value, or combination
- Example
 - `calculatePay()` function

Tracing Errors with the console.log() Method (cont'd.)

```
function calculatePay() {  
    var payRate = 15; numHours = 40;  
    var grossPay = payRate * numHours;  
    console.log("grossPay is " + grossPay);  
    var federalTaxes = grossPay * .06794;  
    var stateTaxes = grossPay * .0476;  
    var socialSecurity = grossPay * .062;  
    var medicare = grossPay * .0145;  
    var netPay = grossPay - federalTaxes;  
    console.log("grossPay minus federalTaxes is " + netPay);  
    netPay *= stateTaxes;  
    console.log("netPay minus stateTaxes is " + netPay);  
    netPay *= socialSecurity;  
    console.log("netPay minus socialSecurity is " + netPay);  
    netPay *= medicare;  
    console.log("netPay minus medicare is " + netPay);  
    return netPay;  
}  
calculatePay();
```

Tracing Errors with the console.log() Method (cont'd.)



The screenshot shows the developer tools console tab labeled "Console". The tabs above it include "Inspector", "Debugger", "Style Editor", and "Profiler". The "Logging" tab is selected. The "JS" tab is highlighted with a blue dot. The console output area contains the following text:

```
11:22:32.519 "grossPay is 600"
11:22:32.520 "grossPay minus federalTaxes is 559.236"
11:22:32.520 "netPay minus stateTaxes is 15971.78016"
11:22:32.520 "netPay minus socialSecurity is 594150.221952"
11:22:32.520 "netPay minus medicare is 5169106.930982401"
```

A black arrow points from the bottom right of the main console area to a callout box containing the same five log statements, which are also enclosed in a border.

Figure 4-11 Contents of the console after executing the calculatePay() function

Tracing Errors with the console.log() Method (cont'd.)

- Driver program
 - simplified, temporary program
 - contains only the code you are testing

Using Comments to Locate Bugs

- Another method of locating bugs
 - “Comment out” problematic lines
- Helps isolate statement causing the error
- When error message first received
 - Start by commenting out only the statement specified by the line number in the error message
 - Continue commenting lines until error eliminated

Combining Debugging Techniques

- Combine debugging techniques
 - Aid in search for errors
- Example:
 - Use comments combined with an alert box or log message to trace errors in the calculatePay() function

Combining Debugging Techniques (cont'd.)

```
function calculatePay() {  
    var payRate = 15;  
    var numHours = 40;  
    var grossPay = payRate * numHours;  
    window.alert(grossPay);  
    // var federalTaxes = grossPay * .06794;  
    // var stateTaxes = grossPay * .0476;  
    // var socialSecurity = grossPay * .062;  
    // var medicare = grossPay * .0145;  
    // var netPay = grossPay - federalTaxes;  
    // netPay *= stateTaxes;  
    // netPay *= socialSecurity;  
    // netPay *= medicare;  
    // return Math.round(netPay);
```

Dependencies

- Relationship in which one statement depends on another statement executing successfully
- Can make debugging more challenging
- Important to retest program after fixing a bug to ensure other parts aren't affected by change

Tracing Errors with Debugging Tools

- Available in current versions of all modern browsers
 - Internet Explorer (IE)
 - Chrome
 - Firefox
- Accessible through same panel that opens when you use the console

Tracing Errors with Debugging Tools (cont'd.)

- Examining code manually
 - Usually first step taken with a logic error
 - Works fine with smaller programs
- Debugging tools
 - Help trace each line of code
 - More efficient method of finding and resolving logic errors

Understanding the IE, Firefox, and Chrome Debugger Windows

- Using Debugger Windows
 - Open a document to debug in a browser
 - Use keyboard shortcut or menu to open debugger

BROWSER	KEYBOARD SHORTCUT	MENU STEPS
Internet Explorer 9+	F12 , then Ctrl + 3	Click the Tools icon, click F12 developer tools on the menu, then in the window that opens, click the Debugger button
Firefox	Ctrl + Shift + S (Win) or option + command + S (Mac)	Click the Open menu button, click Developer , and then click Debugger
Chrome	Ctrl + Shift + J (Win) or option + command + J (Mac), then in the window that opens, click the Sources button	Click the Customize and control Google Chrome button, click Tools , click JavaScript Console , then in the window that opens, click the Sources button

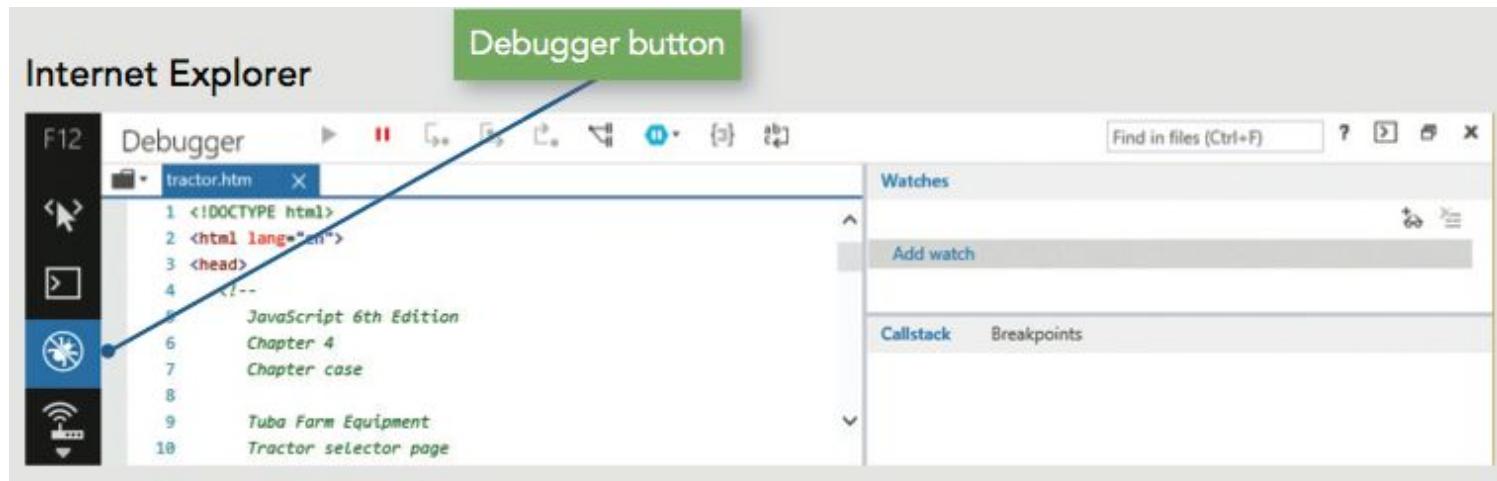
Table 4-1: Steps to open debuggers in IE, Firefox, and Chrome

Understanding the IE, Firefox, and Chrome Debugger Windows (cont'd.)

- Debugger window
 - Usually separate pane attached to bottom of browser window
 - Can also detach pane into separate window

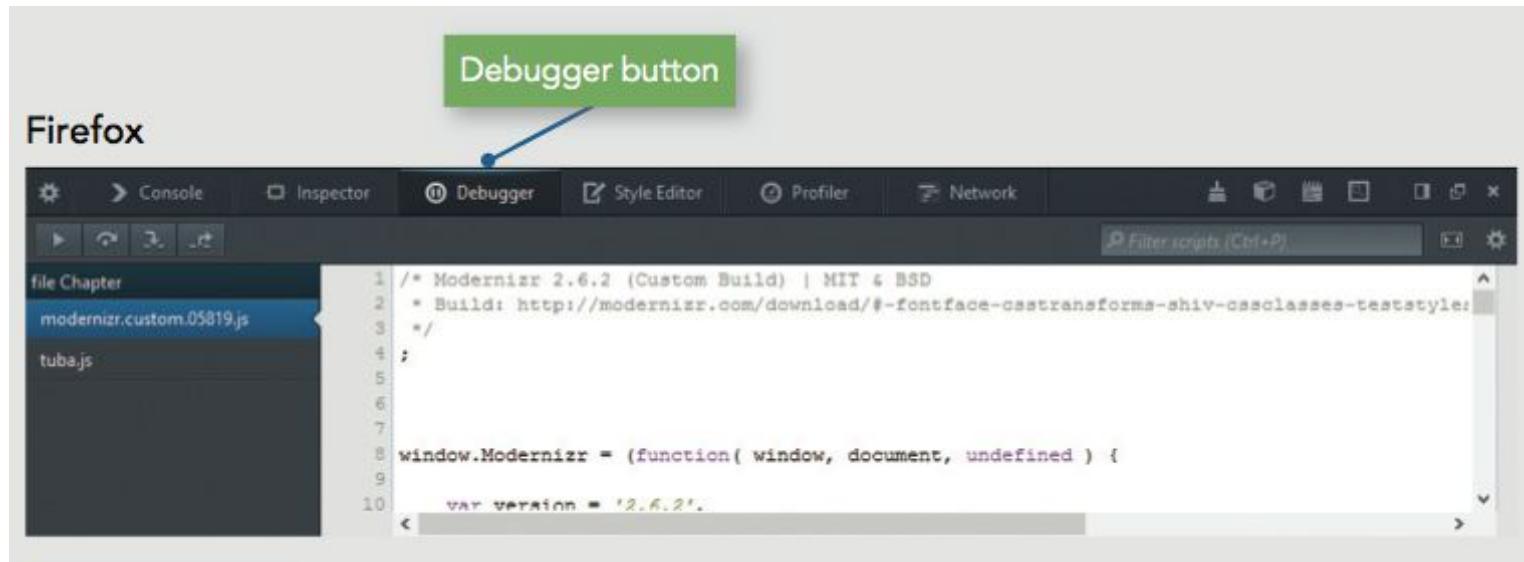
Understanding the IE, Firefox, and Chrome Debugger Windows (cont'd.)

- Internet Explorer
 - Shows HTML code by default
 - Click View sources to select a different file



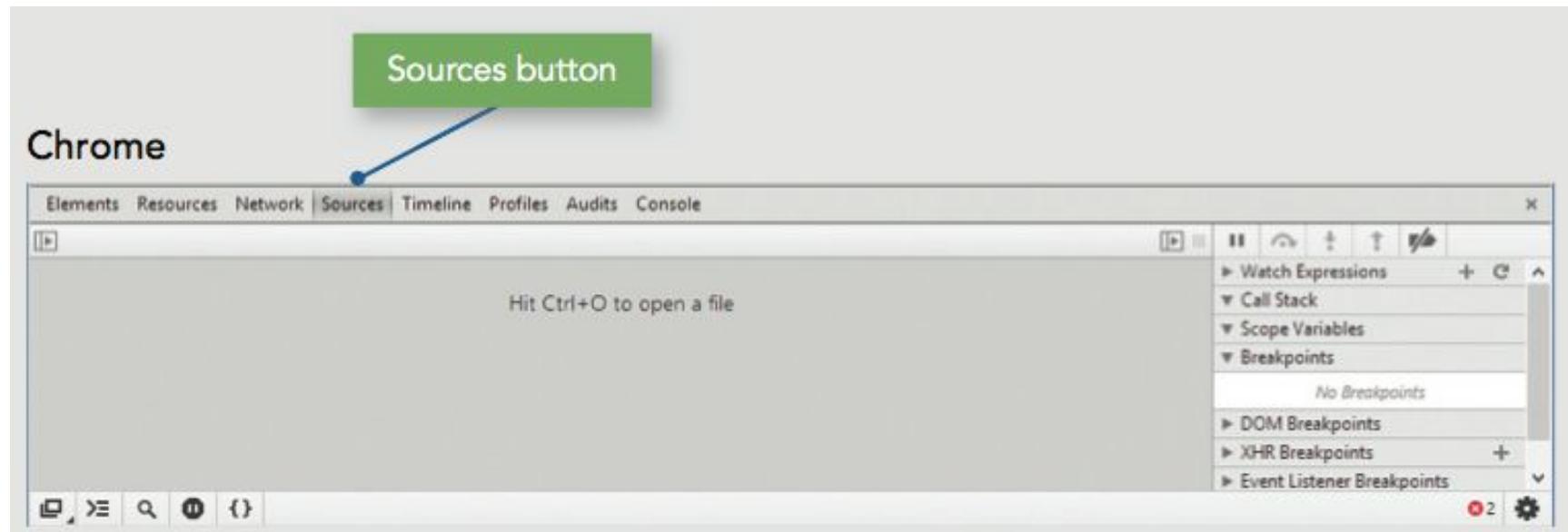
Understanding the IE, Firefox, and Chrome Debugger Windows (cont'd.)

- Firefox
 - Lists JavaScript files alphabetically
 - Click a filename to see its contents



Understanding the IE, Firefox, and Chrome Debugger Windows (cont'd.)

- Chrome
 - Displays no files by default
 - press Ctrl + O (Win) or command + O (Mac) to select from list of associated files



Setting Breakpoints

- Break mode
 - Temporary suspension of program execution
 - Used to monitor values and trace program execution
- Breakpoint
 - Statement where execution enters break mode
- When program paused at a breakpoint
 - Use debug tools to trace program execution

Setting Breakpoints (cont'd.)

- To set a breakpoint
 - Click the line number of the statement where execution should stop
- Resume button (Firefox/Chrome), Continue button (IE)
 - Executes rest of the program normally or until another breakpoint encountered

Setting Breakpoints (cont'd.)

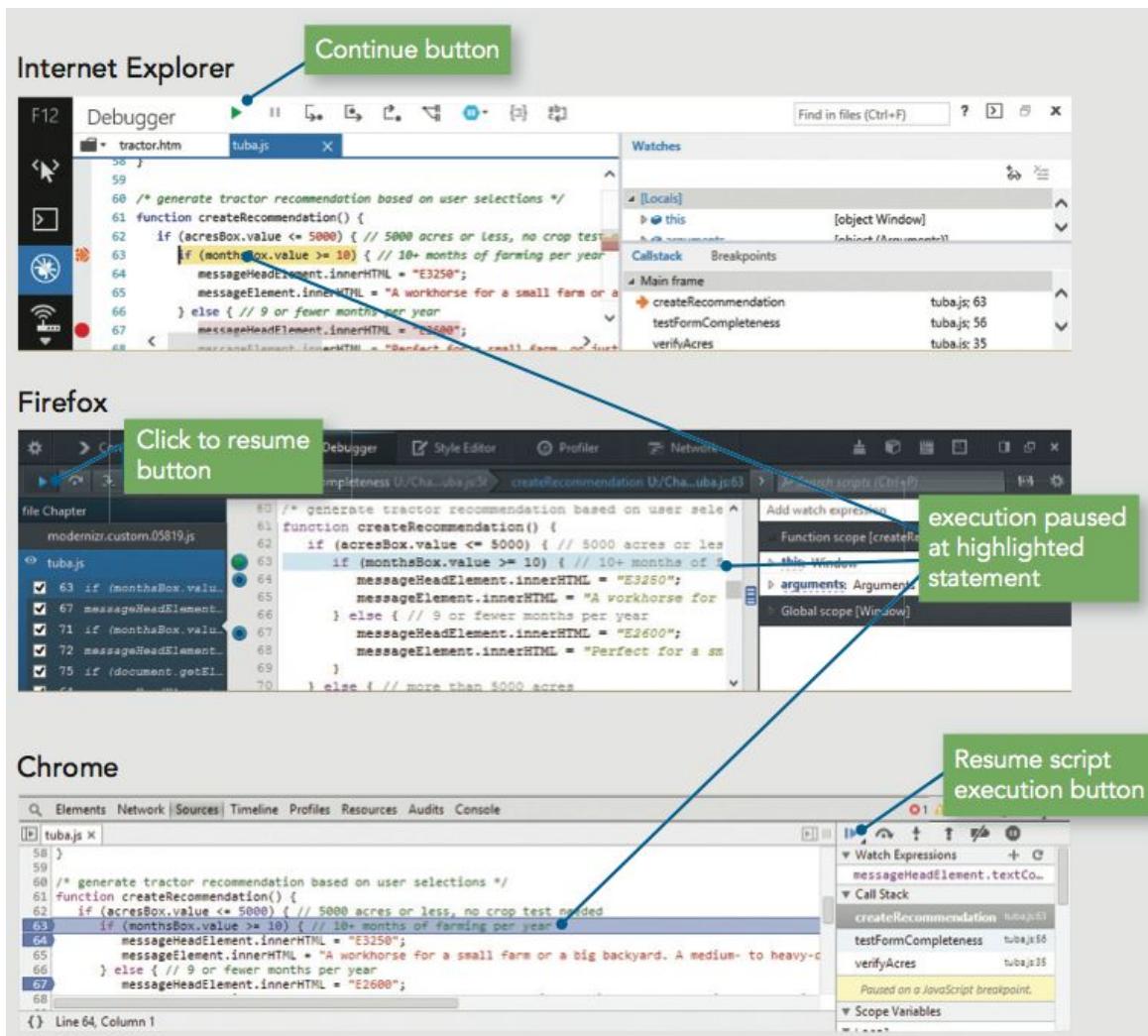


Figure 4-20 tuba.js execution stopped at the line 63 breakpoint

Clearing Breakpoints

- To clear a breakpoint
 - Click the line number
- To clear all breakpoints
 - Right-click any breakpoint
 - Click "Remove all breakpoints" or "Delete all"

Stepping Through Your Scripts

- Stepping into
 - Executes an individual line of code
 - Pauses until instructed to continue
 - Debugger stops at each line within every function
- Stepping over
 - Allows skipping of function calls
 - Program still executes function stepped over
- Stepping out
 - Executes all remaining code in the current function
 - Debugger stops at next statement in the calling function

Tracing Variables and Expressions

- Variables list
 - Displays all local variables within the currently executing function
 - Shows how different values in the currently executing function affect program execution
- Watch list
 - Monitors variables and expressions in break mode

Tracing Variables and Expressions (cont'd.)

- To access watch list
 - IE
 - Displayed by default on right side of pane
 - In break mode, local and global variables displayed
 - Firefox
 - Click Expand Panes button
 - Shows watch and variables list on right side of pane
 - Chrome
 - Displayed by default on right side of pane

Tracing Variables and Expressions (cont'd.)

- To add an expression to the watch list
 - Locate an instance of the expression in the program
 - Select it and copy it to the Clipboard
 - Click "Click to add" (IE) or "Add watch expression (Firefox or Chrome)
 - Paste expression from Clipboard
 - Press Enter

Examining the Call Stack

- Call stack
 - Ordered lists of which procedures (functions, methods, event handlers) have been called but haven't finished executing
- Each time a program calls a procedure:
 - Procedure added to top of the call stack

Examining the Call Stack

- IE and Chrome
 - Call stack list displayed to right of code
- Firefox
 - Call stack list displayed above code

Handling Exceptions and Errors

- Bulletproofing
 - Writing code to anticipate and handle potential problems
- One bulletproofing technique
 - Validate submitted form data
- Exception handling
 - Allows programs to handle errors as they occur in program execution
- Exception
 - Error occurring in a program

Throwing Exceptions

- Execute code containing an exception in a `try` statement
- `throw` statement
 - Specifies an error message in case an error that occurs within a `try` block

```
try {  
    var lastName = document.getElementById("lName").value;  
    if (lastName === "") {  
        throw "Please enter your last name."  
    }  
}
```

Catching Exceptions

- Use a `catch` statement
 - Handles, or “catches” the error

- Syntax:

```
catch(error) {  
    statements;  
}
```

- Example:

```
catch(lNameError) {  
    window.alert(lNameError);  
    return false;  
}
```

Executing Final Exception Handling Tasks

- `finally` statement
 - Executes regardless of whether its associated `try` block throws an exception
 - Used to perform some type of cleanup
 - Or any necessary tasks after code evaluated with a `try` statement

Implementing Custom Error Handling

- Primary purpose of exception handling
 - Prevent users from seeing errors occurring in programs
 - Provide graceful way to handle errors
- Reason for using exception handling with JavaScript
 - Evaluate user input
- Programmers may write their own error-handling code
 - Can write user-friendly messages
 - Provides greater control over any errors

Implementing Custom Error Handling (cont'd.)

- Catching errors with the `error` event
 - Executes whenever error occurs on a Web page
 - Name of function to handle JavaScript errors
 - Assigned as event listener for `error` event
 - Preventing the Web browser from executing its own error handling functionality
 - Return `return a value of true` from the `error` event handler function

Implementing Custom Error Handling (cont'd.)

- Writing custom error-handling functions
 - JavaScript interpreter automatically passes three arguments to the custom error handling function
 - Error message, URL, line number
 - Use these values in custom error handling function
 - By adding parameters to the function definition
 - Use parameters in the function
 - Show a user the location of any JavaScript errors that may occur

Additional Debugging Techniques

- Includes
 - Checking HTML elements
 - Analyzing logic
 - Testing statements with console command line
 - Using the debugger statement
 - Executing code in strict mode
 - Linting
 - Reloading a Web page

Checking HTML Elements

- If a bug cannot be located using methods described in this chapter:
 - Perform a line-by-line analysis of the HTML code
 - Ensure all necessary opening and closing tags included
- Use code editor specialized for web development
 - Highlights syntax errors as you type
- Use the W3C Markup Validation Service to validate a Web page

Analyzing Logic

- Some JavaScript code errors stem from logic problems
 - Can be difficult to spot using tracing techniques
- Analyze each statement on a case-by-case basis

Testing Statements with the Console Command Line

- Console command line
 - Testing and executing JavaScript statements
 - Without HTML document or JavaScript source file
 - Useful if trying to construct the correct syntax for a mathematical expression
- Enter JavaScript statement at command line in web browser's console
- Including multiple statements at the command line
 - Separate statements with a semicolon

Using the debugger statement

- When you include the `debugger` statement in your code
 - web browser stops executing JavaScript code when it reaches the `debugger` statement
 - equivalent of a breakpoint that's part of your JavaScript code

Using Strict Mode

- Strict mode
 - Removes some features from JavaScript
 - Requires more stringent syntax for other features
 - Example: must always use `var` to declare variables
- Many removed or altered features in strict mode are known to cause hard to find bugs
- Include statement `"use strict";`
 - Including at start of script section requests strict mode for all code in that section
 - Including at start of code block in function requests strict mode just for that function

Linting

- Running code through a program that flags some common issues that may affect code quality
- jslint is a commonly used linting program
- Similar result to using strict mode, but generates a report containing line numbers

Reloading a Web Page

- Usually click the browser Reload or Refresh button
- Web browser cannot always completely clear its memory
 - Remnants of an old bug may remain
 - Force web page reload
 - Hold Shift key and click the browser's Reload or Refresh button
- May need to close browser window completely
- May need to delete frequently visited web pages

Summary

- Three types of program errors
 - Syntax errors, run-time errors, logic errors
 - Error messages
 - First line of defense in locating bugs
- Tracing
 - Examination of individual statements in an executing program
- Using `console.log()` method to trace bugs
 - Helpful to use a driver program
- Browser debugging tools

Summary (cont'd.)

- Break mode
 - Temporary suspension of execution to monitor values and trace execution
- Breakpoint: statement in the code at which program execution enters break mode
- Stepping into, stepping over, and stepping out
- Variables list and watch list
- Call stack
 - List of procedures that have started but not finished

Summary (cont'd.)

- Bulletproofing
 - Writing code to anticipate, handle potential problems
- Exception handling
- try, throw, catch, finally statements
- JavaScript includes an error event
 - Executes whenever an error occurs on a web page
- Additional debugging methods and techniques
 - Checking HTML elements, analyzing logic, console command line, debugger statement, strict mode, linting, and reloading a web page

Javascript

Chapter 5
*Working with the Document Object
Model (DOM) and DHTML*

Objectives

When you complete this chapter, you will be able to:

- Access elements by id, tag name, class, name, or selector
- Access element content, CSS properties, and attributes
- Add and remove document nodes
- Create and close new browser tabs and windows with an app

Objectives (cont'd.)

When you complete this chapter, you will be able to:

- Use the `setTimeout()` and `setInterval()` methods to specify a delay or a duration
- Use the History, Location, Navigation, and Screen objects to manipulate the browser window

Understanding the Browser Object Model and the Document Object Model

- JavaScript treats web page content as set of related components
 - objects
- Every element on web page is an object
- You can also create objects
 - a function is an object

Understanding the Browser Object Model

- Browser object model (BOM) or client-side object model
 - Hierarchy of objects
 - Each provides programmatic access
 - To a different aspect of the web browser window or the web page
- Window object
 - Represents a Web browser window
 - Called the global object
 - Because all other BOM objects contained within it

Understanding the Browser Object Model (cont'd.)

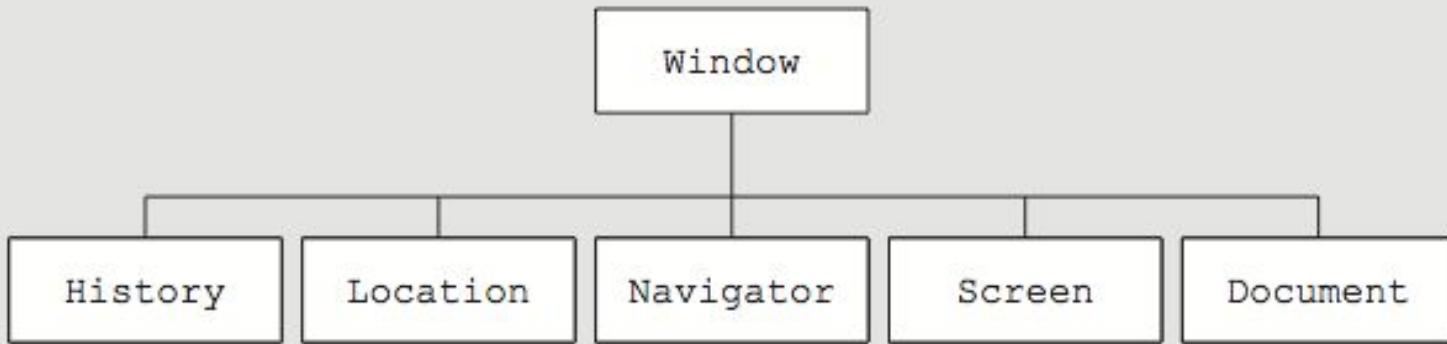


Figure 5-3 Browser object model

The Document Object Model

- Document object
 - Represents the Web page displayed in a browser
 - Contains all Web page elements
 - JavaScript represents each element by its own object

The DOM and DHTML

- Dynamic HTML (DHTML)
 - Interaction can change content of web page without reloading
 - Can also change presentation of content
 - Combination of HTML, CSS, and JavaScript
- DOM
 - example of an application programming interface (API)
 - structure of objects with set of properties and methods

The DOM tree

- The DOM hierarchy depends on a document's contents

```
1  <html lang="en">
2      <head>
3          <meta charset="utf-8" />
4          <title>Photo Gallery</title>
5      </head>
6      <body>
7          <header>
8              <h1>Garden Photo</h1>
9          </header>
10         <article>
11             <figure>
12                 <figcaption>Butterfly bush</figcaption>
13                 
14             </figure>
15         </article>
16     </body>
17 </html>
```

The DOM tree (cont'd.)

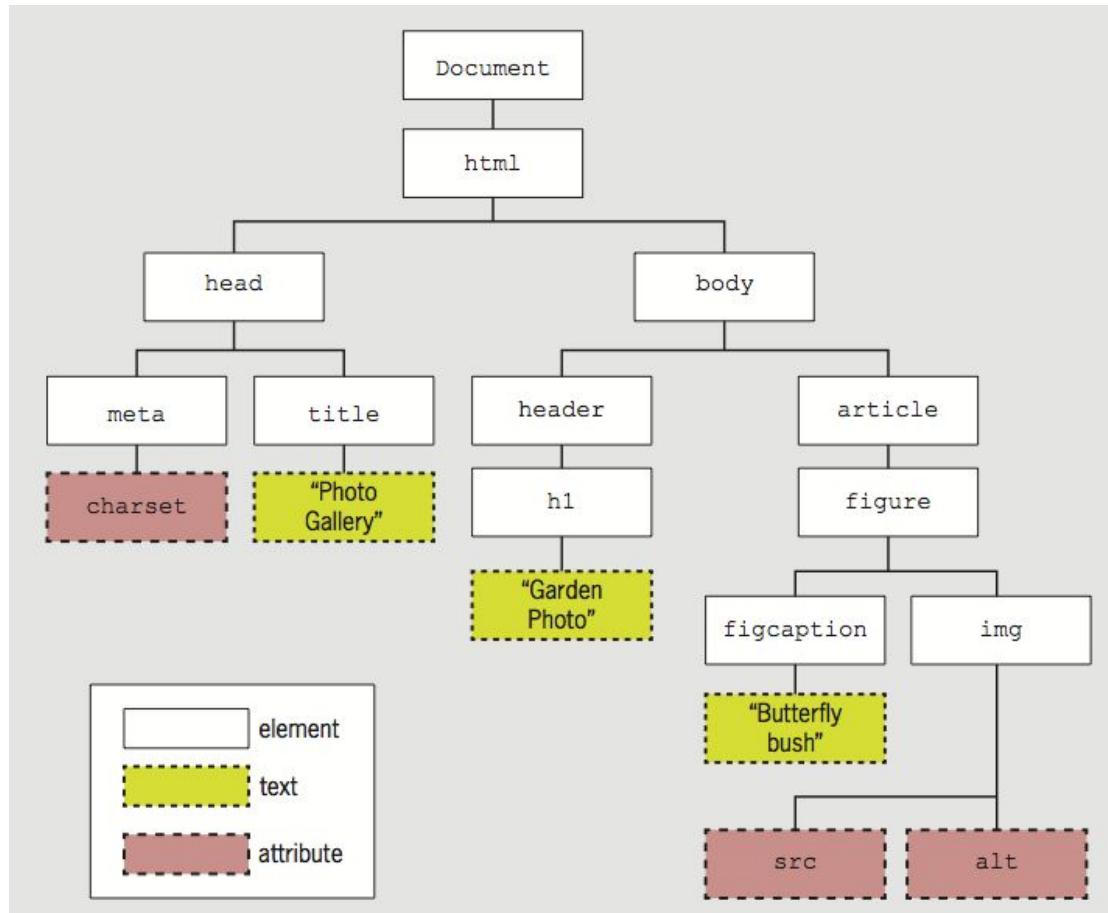


Figure 5-4 Example DOM tree

The DOM tree

- Each item in the DOM tree is a node
- Element, attribute, and text content nodes are most commonly used

DOM Document Object Methods

METHOD	DESCRIPTION
<code>getElementById(<i>ID</i>)</code>	Returns the element with the <code>id</code> value <i>ID</i>
<code>getElementsByClassName(<i>class1</i> [<i>class2</i> ...])</code>	If one class name, <code>class1</code> , is specified, returns the collection of elements that belong to <code>class1</code> ; if two or more space-separated class names are specified, the returned collection consists of those elements that belong to all specified class names
<code>getElementsByName(<i>name</i>)</code>	Returns the collection of elements with the name <i>name</i>
<code>getElementsByTagName(<i>tag</i>)</code>	Returns the collection of elements with the tag (element) name <i>tag</i>
<code>querySelectorAll(<i>selector</i>)</code>	Returns the collection of elements that match the CSS selector specified by <i>selector</i>
<code>write(<i>text</i>)</code>	Writes <i>text</i> to the document

Table 5-1 HTML DOM Document object methods

DOM Document Object Properties

PROPERTY	DESCRIPTION
body	Document's <code>body</code> element
cookie	Current document's cookie string, which contains small pieces of information about a user that are stored by a web server in text files on the user's computer
domain	Domain name of the server where the current document is located
lastModified	Date the document was last modified
location	Location of the current document, including its URL
referrer	URL of the document that provided a link to the current document
title	Title of the document as specified by the <code>title</code> element in the document head section
URL	URL of the current document

Table 5-2 Selected DOM Document object properties

Accessing Document Elements, Content, Properties, and Attributes

- Methods such as `getElementById()` are methods of the `Document` object
- Several methods available for JavaScript to reference web page elements

Accessing Elements by id value

- Set the id value in HTML
- getElementById() method
 - Returns the first element in a document with a matching id attribute
- Example:

HTML element with id value

```
<input type="number" id="zip" />
```

JavaScript to reference HTML element

```
var zipField = document.getElementById("zip");
```

Accessing Elements by Tag Name

- `getElementsByName()` method
 - Returns array of elements matching a specified tag name
 - Tag name is name of element
- Method returns a set of elements
 - Node list is indexed collection of nodes
 - HTML collection is indexed collection of HTML elements
 - Either set uses array syntax

Accessing Elements by Tag Name (cont'd.)

- Example:
 - Index numbers start at 0, so second element uses index number 1
 - To work with the *second* h1 element on a page:

```
var secondH1 = document.getElementsByTagName("h1")[1];
```

Accessing Elements by Class Name

- `getElementsByClassName()` method
 - Returns node list or HTML collection of elements with a `class` attribute matching a specified value
- Example
 - All elements with `class` value `side`:

```
var sideElements = document.getElementsByClassName("side");
```

Accessing Elements by Class Name (cont'd.)

- class attribute takes multiple values, so getElementsByClassName () method takes multiple arguments
- Arguments enclosed in single set of quotes, with class names separated by spaces
- Example
 - All elements with class values side and green:

```
var sideGreenElements = document.getElementsByClassName("side green");
```

Accessing Elements by Name

- `getElementsByName()` method
 - Returns node list or HTML collection of elements with a `name` attribute matching a specified value
- Not as useful as preceding options
 - But creates more concise code when accessing set of option buttons or check boxes in a form:

```
var colorButtons = document.getElementsByName("color");
```
- Not standard in IE9 and earlier versions of IE, so important to test

Accessing Elements with CSS Selectors

- `querySelector()` method
 - References elements using CSS syntax
 - Returns first occurrence of element matching a CSS selector
- Example:

HTML:

```
<header>  
    <h1></h1>
```

JavaScript to reference `img` element

```
</header>
```

```
querySelector("header h1 img")
```

Accessing Elements with CSS Selectors (cont'd)

- IE8 supports only simple selectors
 - Can use different approach in creating CSS selector
 - Previous example could be rewritten as

```
querySelector("img.logo")
```

Accessing Elements with CSS Selectors (cont'd)

- `querySelectorAll()` method
 - Returns collection of elements matching selector
 - Different from `querySelector()` method, which returns only first occurrence
 - Example: HTML: `<nav>`

```
<ul>  
  <li>About Us</li>  
  <li>Order</li>  
  <li>Support</li>
```

JavaScript to reference all three `li` elements:

```
querySelectorAll("nav ul li")
```

Accessing an Element's Content

- `textContent` property
 - Accesses and changes text that an element contains
 - Unlike `innerHTML`, `textContent` strips out HTML tags
- Example:

HTML:

```
<ul>
    <li class="topnav"><a href="aboutus.htm">About Us</a></li>
    <li class="topnav"><a href="order.htm">Order</a></li>
    <li class="topnav"><a href="support.htm">Support</a></li>
</ul>
```

JavaScript to reference and access first li element:

```
var button1 = querySelectorAll("li.topNav") [0];
var allContent = button1.innerHTML;
// <a href="aboutus.htm">About Us</a>
var justText = button1.textContent;
// About Us
```

Accessing an Element's Content (cont'd)

- `textContent` property is more secure
 - Not supported by IE8 or earlier
 - Some developers use `if/else` construction to implement `textContent` only on supported browsers

Accessing an Element's CSS Properties

- Can access CSS properties through DOM
 - Use dot notation
 - Reference element's style property followed by name of CSS property
 - Example: change value of CSS display property to none for element with id value logo:

```
document.getElementById("logo").style.display = "none";
```

Accessing an Element's CSS Properties (cont'd.)

- When CSS property includes hyphen (-), remove hyphen and capitalize letter following hyphen
 - Use dot notation
 - `font-family` becomes `fontFamily`
 - Example:

```
var font = document.getElementById("logo").style.fontFamily;
```

- CSS value specified using DOM reference is an inline style
 - Higher priority than embedded or external styles

Accessing an Element's CSS Properties (cont'd.)

- To remove a style you previously added with a DOM reference
 - set its value to an empty string
 - Example:

```
document.getElementById("navbar").style.color = "";
```

Accessing Element Attributes

- Access element attribute with period and name of attribute after element reference
 - Reference element with `id` value `homeLink`:
`document.getElementById("homeLink")`
 - Reference `href` attribute of same element:
`document.getElementById("homeLink").href`
- Can use to look up attribute value and assign to variable, or to assign new value to attribute

Accessing Element Attributes (cont'd)

- One exception for accessing element attributes
 - Must use property name `className` to refer to `class` attribute values
 - Single class value returned like standard attribute value
 - Multiple class values returned in single string, separated by spaces

Adding and Removing Document Nodes

- DOM includes methods to change DOM tree
 - Can create brand new elements
 - Can add/remove elements from DOM tree

Creating Nodes

- createElement () method

- Creates a new element

- Syntax:

- document.createElement("element")

- *element* is an element name

- Example:

- To create a new div element:

```
document.createElement("div");
```

Attaching Nodes

- Newly created node is independent of DOM tree
- `appendChild()` method
 - Attaches node to DOM tree
 - Syntax:
parentNode.appendChild(childNode)
 - *childNode* is node to be attached
 - *parentNode* is node to attach child node to

Attaching Nodes (cont'd.)

- Example:
 - Create new `li` element and attach to element with `id` value `navList`:

```
var list = document.getElementById("navList");  
var contact = document.createElement("li");  
list.appendChild(contact);
```
- Document fragment
 - Set of connected nodes not part of document
 - Can use `appendChild()` to add document fragment to DOM tree for a document

Attaching Nodes (cont'd.)

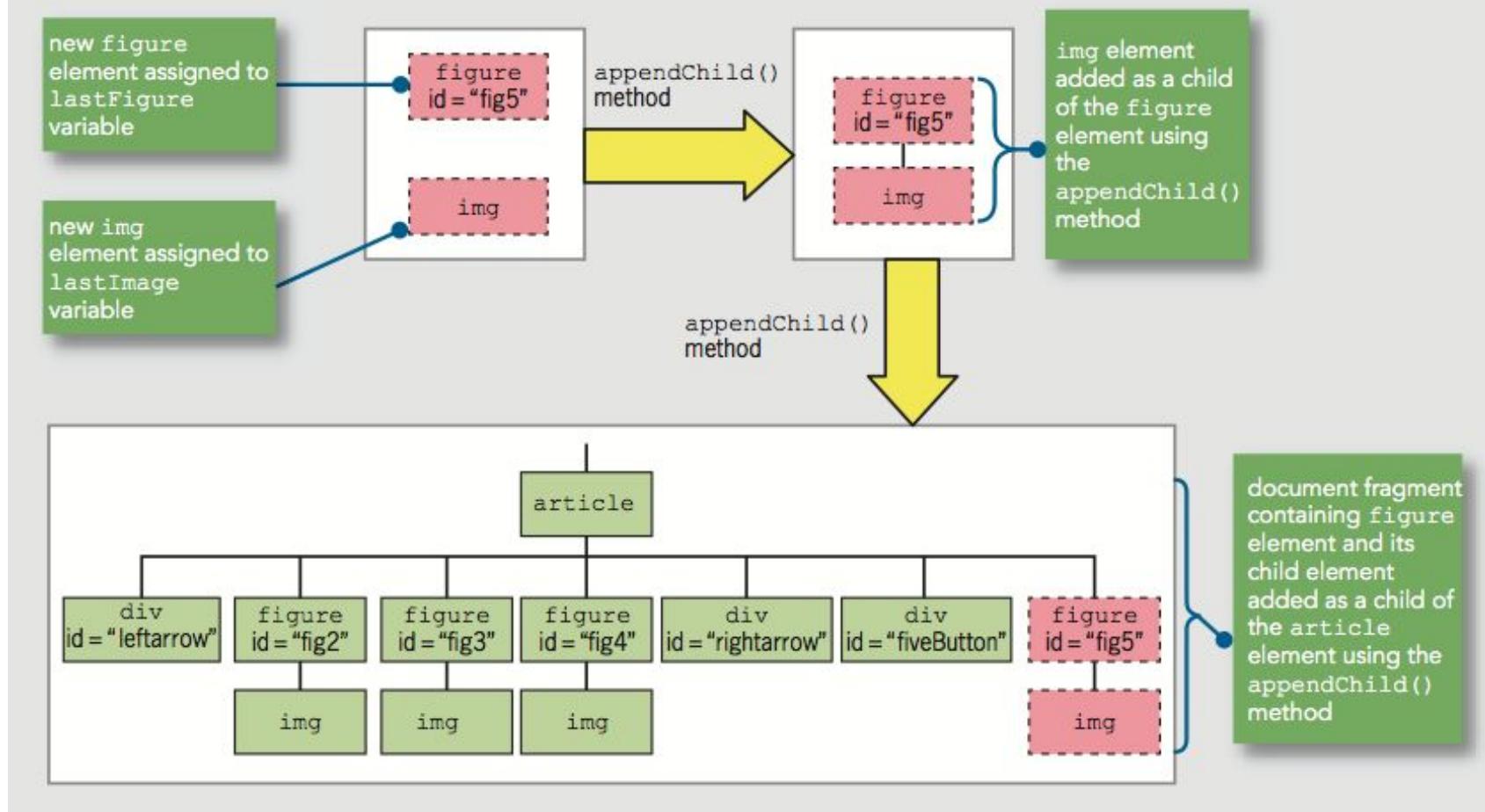


Table 5-7 Using the `appendChild()` method to attach nodes

Cloning Nodes

- Create new node same as existing node
- `cloneNode()` method
- Syntax:
 - `existingNode.cloneNode(true | false)`
 - `true` argument clones child nodes
 - `false` argument clones only specified parent node
- Example:

```
document.createElement("div");
```

Cloning Nodes (cont'd.)

- Example:

```
var contact = document.createElement("li");  
contact.className = "mainNav";  
contact
```

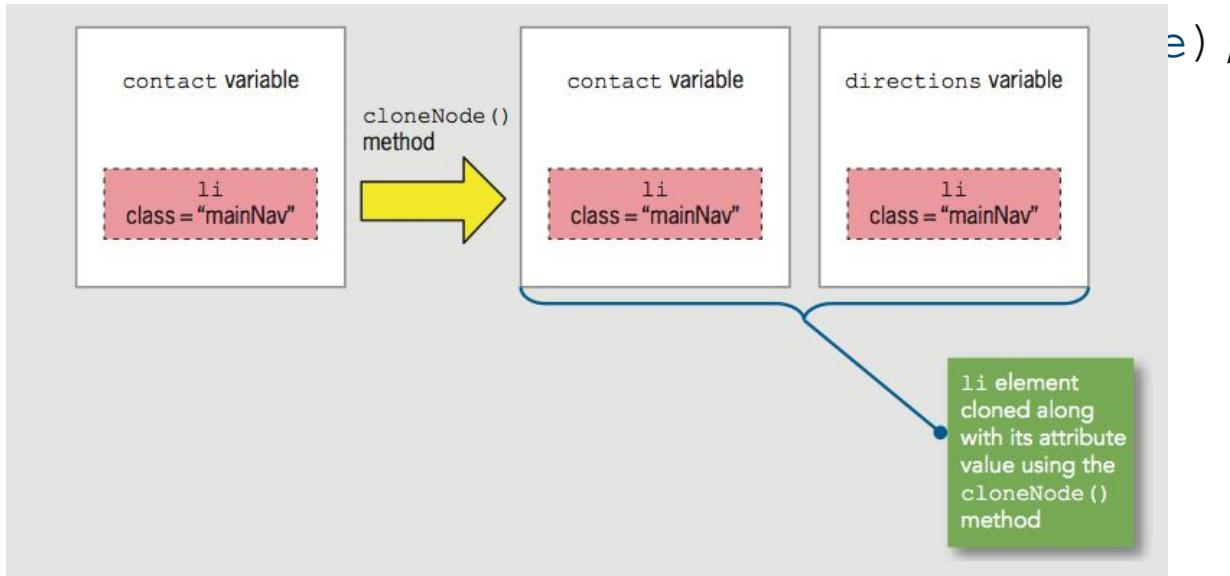


Figure 5-10 Using the `cloneNode ()` method

Inserting Nodes at Specific Positions in the Document Tree

- New node created with `createElement()` is not attached to document tree
- `appendChild()` adds node after existing child nodes
- To specify a different position, use `insertBefore()`
- **Syntax:**
 - `parentNode.insertBefore(newChildNode, existingChildNode)`

Inserting Nodes at Specific Positions in the Document Tree (cont'd.)

- Example:

- HTML:

```
<ul id="topnav">  
    <li><a href="aboutus.htm">About Us</a></li>  
    <li><a href="order.htm">Order</a></li>
```

- JavaScript:

```
</ul>  
var list = document.getElementById("topnav") ;  
var directions = document.createElement("li") ;  
directions.innerHTML = "Directions" ;  
var aboutus = document.querySelectorAll("#topnav li") [0] ;  
list.insertBefore(directions, aboutus) ;
```

Inserting Nodes at Specific Positions in the Document Tree (cont'd.)

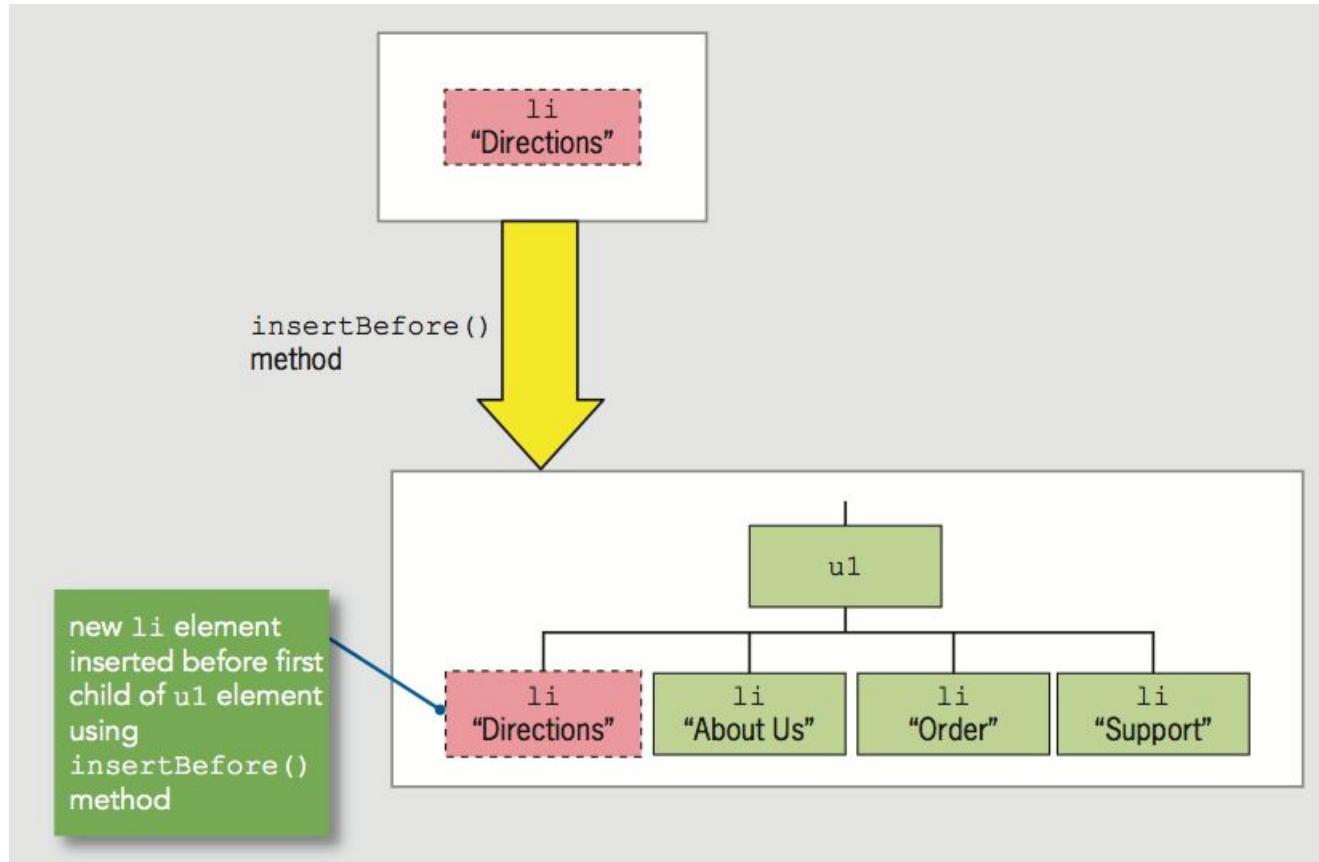


Figure 5-14 Using the `insertBefore()` method

Removing Nodes

- `removeNode()` removes node from DOM tree
- **Syntax:**

— `parentNode.removeChild(childNode)`

- Can assign removed node to variable:

```
var list = document.getElementById("topnav");  
  
var aboutus = document.querySelectorAll("#topnav li")[0];  
  
var aboutNode = list.removeChild(aboutus);
```

- Node removed without being assigned to a variable is deleted during garbage collection

Manipulating the Browser with the Window Object

- Window object
 - Includes properties containing information about the web browser window or tab
 - Contains methods to manipulate the web browser window or tab itself

Manipulating the Browser with the Window Object (cont'd.)

PROPERTY	DESCRIPTION
<code>closed</code>	Boolean value that indicates whether a window or tab has been closed
<code>document</code>	Reference to the <code>Document</code> object
<code>history</code>	Reference to the <code>History</code> object
<code>innerHeight</code>	Height of the window area that displays content, including the scrollbar if present
<code>innerWidth</code>	Width of the window area that displays content, including the scrollbar if present
<code>location</code>	Reference to the <code>Location</code> object
<code>name</code>	Name of the window or tab
<code>navigator</code>	Reference to the <code>Navigator</code> object
<code>opener</code>	Reference to the window that opened the current window or tab
<code>outerHeight</code>	Height of the entire browser window
<code>outerWidth</code>	Width of the entire browser window
<code>screen</code>	Reference to the <code>Screen</code> object
<code>self</code>	Self-reference to the <code>Window</code> object; identical to the <code>window</code> property
<code>status</code>	Temporary text that is written to the status bar
<code>window</code>	Self-reference to the <code>Window</code> object; identical to the <code>self</code> property

Table 5-3 Window object properties

Manipulating the Browser with the Window Object (cont'd.)

METHOD	DESCRIPTION
<code>alert()</code>	Displays a simple message dialog box with an OK button
<code>blur()</code>	Removes focus from a window or tab
<code>clearInterval()</code>	Cancels an interval that was set with <code>setInterval()</code>
<code>clearTimeout()</code>	Cancels a timeout that was set with <code>setTimeout()</code>
<code>close()</code>	Closes a web browser window or tab
<code>confirm()</code>	Displays a confirmation dialog box with OK and Cancel buttons
<code>focus()</code>	Makes a <code>window</code> object the active window or tab
<code>moveBy()</code>	Moves the window relative to the current position
<code>moveTo()</code>	Moves the window to an absolute position
<code>open()</code>	Opens a new web browser window or tab
<code>print()</code>	Prints the document displayed in the current window or tab
<code>prompt()</code>	Displays a dialog box prompting a user to enter information

Table 5-4 `window` object methods (*continues*)

Manipulating the Browser with the Window Object (cont'd.)

METHOD	DESCRIPTION
<code>resizeBy()</code>	Resizes a window by a specified amount
<code>resizeTo()</code>	Resizes a window to a specified size
<code>scrollBy()</code>	Scrolls the window or tab by a specified amount
<code>scrollTo()</code>	Scrolls the window or tab to a specified position
<code>setInterval()</code>	Repeatedly executes a function after a specified number of milliseconds have elapsed
<code>setTimeout()</code>	Executes a function once after a specified number of milliseconds have elapsed

Table 5-4 Window object methods

Manipulating the Browser with the Window Object (cont'd.)

- self property
 - Refers to the current Window object
 - Identical to using the window property to refer to the Window object
 - Examples:
 - `window.close();`
 - `self.close();`
- Web browser assumes reference to global object
- Good practice
 - Use window or self references
 - When referring to a Window object property or method

Opening and Closing Windows

- Reasons to open a new Web browser window
 - To launch a new Web page in a separate window
 - To use an additional window to display information
- When new Web browser window opened:
 - New Window object created
 - Represents the new window
- Know how to open a link in a new window using the a element's target attribute

```
<a href="http://www.wikipedia.org/">  
    target="wikiWindow">Wikipedia home page</a>
```

Opening a Window

- open () method of the Window object
 - Opens new windows
- Syntax

```
window.open(url, name, options, replace);
```

METHOD	DESCRIPTION
resizeBy()	Resizes a window by a specified amount
resizeTo()	Resizes a window to a specified size
scrollBy()	Scrolls the window or tab by a specified amount
scrollTo()	Scrolls the window or tab to a specified position
setInterval()	Repeatedly executes a function after a specified number of milliseconds have elapsed
setTimeout()	Executes a function once after a specified number of milliseconds have elapsed

Table 5-5 Arguments of the Window object's open () method

Opening a Window (cont'd.)

- Include all (or none) `window.open()` method arguments
- Example:
 - `window.open("http://www.wikipedia.org");`

Opening a Window (cont'd.)

- Customize new browser window or tab appearance
 - Use `window.open()` method options argument

NAME	DESCRIPTION
<code>height</code>	Sets the window's height
<code>left</code>	Sets the horizontal coordinate of the left of the window, in pixels
<code>location</code>	Includes the URL Location text box
<code>menubar</code>	Includes the menu bar
<code>personalbar</code>	Includes the bookmarks bar (or other user-customizable bar)
<code>resizable</code>	Determines if the new window can be resized
<code>scrollbars</code>	Includes scroll bars
<code>status</code>	Includes the status bar
<code>toolbar</code>	Includes the Standard toolbar
<code>top</code>	Sets the vertical coordinate of the top of the window, in pixels
<code>width</code>	Sets the window's width

Table 5-6 Common options of the `Window` object's `open()` method

Opening a Window (cont'd.)

- `window.open()` method name **argument**
 - Same as value assigned to the `target` attribute
 - Specifies window name where the URL should open
 - If name argument already in use
 - JavaScript changes focus to the existing Web browser window instead of creating a new window

Opening a Window (cont'd.)

- Window object's name property used to specify a target window with a link
 - Cannot be used in JavaScript code
- Assign the new Window object created with the `window.open()` method to a variable to control it
- `focus()` method
 - Makes a window the active window

Closing a Window

- `close()` method
 - Closes a web browser window
- `window.close()` or `self.close()`
 - Closes the current window

Working with Timeouts and Intervals

- Window object's timeout and interval methods
 - Creates code that executes automatically
- setTimeout () method
 - Executes code after a specific amount of time
 - Executes only once
 - Syntax
 - `var variable = setTimeout("code", milliseconds);`
- clearTimeout () method
 - Cancel setTimeout () before its code executes
- Example on next slide

Working with Timeouts and Intervals (cont'd.)

```
var buttonNotPressed = setTimeout("window.alert('Your changes have been saved')", 10000);

function buttonPressed() {
    clearTimeout(buttonNotPressed);
    window.open(index.htm);
}
```

Working with Timeouts and Intervals (cont'd.)

- `setInterval()` method
 - Repeatedly executes the same code after being called only once
 - Syntax:
 - `var variable = setInterval("code", milliseconds);`
- `clearInterval()` method
 - Used to clear `setInterval()` method call

The History Object

- History object
 - Maintains internal list (history list)
 - All documents opened during current web browser session
- Security features
 - Will not display URLs contained in the history list

The History Object (cont'd.)

METHOD	DESCRIPTION
<code>back()</code>	Produces the same result as clicking a browser's Back button
<code>forward()</code>	Produces the same result as clicking a browser's Forward button
<code>go()</code>	Opens a specific document in the history list

Table 5-7 Methods of the History object

The History Object (cont'd.)

- `go()` method
 - Allows navigation to a specific previously visited web page
- History object `length` property
 - Provides specific number of documents opened during the current browser session
 - Example:
 - Return to first document opened in current browser session:

```
history.go( - (history.length - 1) );
```

The Location Object

- Location object
 - Allows changes to a new web page from within JavaScript code
- Location object properties allow modification of URL individual portions
 - Web browser automatically attempts to open that new URL

The Location Object (cont'd.)

PROPERTIES	DESCRIPTION
hash	URL's anchor
host	Host and domain name (or IP address) of a network host
hostname	Combination of the URL's host name and port sections
href	Full URL address
pathname	URL's path
port	URL's port
protocol	URL's protocol
search	URL's search or query portion

Table 5-8 Properties of the Location object

METHOD	DESCRIPTION
assign()	Loads a new web page
reload()	Causes the page that currently appears in the web browser to open again
replace()	Replaces the currently loaded URL with a different one

Table 5-9 Methods of the Location object

The Location Object (cont'd.)

- Location object's `assign()` method
 - Same action as changing the `href` property
 - Loads a new web page
- Location object's `reload()` method
 - Equivalent to the browser Reload or Refresh button
 - Causes current page to open again
- Location object's `replace()` method
 - Replaces currently loaded URL with a different one

The Navigator Object

- Navigator object
 - Obtains information about current web browser
 - Example: determine type of web browser running

PROPERTIES	DESCRIPTION
appName	Name of the web browser displaying the page
appVersion	Version of the web browser displaying the page
geolocation	API for accessing the user's current location and user permission settings denying or allowing access to that information
onLine	Whether the browser currently has a network connection
platform	Operating system in use on the client computer
userAgent	String stored in the HTTP user-agent request header, which contains information about the browser, the platform name, and compatibility

Table 5-10 Properties of the Navigator object

The Navigator Object (cont'd.)

```
console.log("Web browser name: " + navigator.appName);  
  
console.log("Web browser version: " + navigator.appVersion);  
  
console.log("Operating platform: " + navigator.platform);  
  
console.log("User agent: " + navigator.userAgent);
```



The screenshot shows the Firefox Developer Tools interface with the 'Console' tab selected. The top navigation bar includes tabs for 'Console', 'Inspector', 'Debugger', 'Style Editor', 'Profiler', and 'Network'. Below the tabs are dropdown menus for 'Net', 'CSS', 'JS', 'Security', 'Logging', and a 'Clear' button. A 'Filter output' search bar is also present. The main console area displays the following log entries:

Time	Message	File
12:05:35.131	"Web browser name: Netscape"	example.htm:38
12:05:35.131	"Web browser version: 5.0 (Windows)"	example.htm:39
12:05:35.131	"Operating platform: Win32"	example.htm:40
12:05:35.133	"User agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:27.0) Gecko/20100101 Firefox/27.0"	example.htm:41

Figure 5-19 Navigator object properties in Firefox console

The Screen Object

- Screen object
 - Obtains information about display screen's size, resolution, color depth
- Common use of Screen object properties
 - Centering a web browser window in the middle of the display area

The Screen Object (cont'd.)

PROPERTIES	DESCRIPTION
availHeight	Height of the display screen, not including operating system features such as the Windows taskbar
availWidth	Width of the display screen, not including operating system features such as the Windows taskbar
colorDepth	Display screen's bit depth if a color palette is in use; if a color palette is not in use, returns the value of the <code>pixelDepth</code> property
height	Height of the display screen
pixelDepth	Display screen's color resolution in bits per pixel
width	Width of the display screen

Table 5-11 Properties of the Screen object

The Screen Object (cont'd.)

- Common Screen object properties uses
 - Center a web browser window
 - Example:

```
var winWidth = 300;
```

```
var winHeight = 200;
```

```
var leftPosition = (screen.width - winWidth) / 2;
```

```
var topPosition = (screen.height - winHeight) / 2;
```

```
var optionString = "width=" + winWidth + ",height=" +  
    + winHeight + ",left=" + leftPosition + ",top=" +  
    + topPosition;
```

Summary

- Browser object model (BOM) or client-side object model
 - Hierarchy of objects
- Top-level object in the browser object model
 - Window object
- Document object: most important object
- DOM represents web page displayed in window

Summary (cont'd.)

- Access elements with `getElementById()`,
`getElementsByTagName()`,
`getElementsByClassName()`,
`getElementsByName`, `querySelector()`, or
`querySelectorAll()`
- Access element content with `textContent()` or
`innerHTML()` property
- Access CSS properties using an element's
JavaScript `style` property

Summary (cont'd.)

- Create new node with `createElement()`
- Attach node with `appendChild()`
- Clone node with `cloneNode()`
- Attach node in specific place with
`insertBefore()`
- Remove node with `removeNode()`
- Open new window with `window.open()`
- Close window with `window.close()`

Summary (cont'd.)

- `setTimeout()` **executes code after specific amount of time**
- `clearTimeout()` **cancels** `setTimeout()`
- `setInterval()` **executes code repeatedly**
- `clearInterval()` **cancels** `setInterval()`

Summary (cont'd.)

- History object maintains an opened documents history list
- Location object allows changes to a new web page from within JavaScript code
- Navigator object obtains information about the current web browser
- Screen object obtains information about the display screen's size, resolution, color depth

JavaScript

Chapter 6
Enhancing and Validating Forms

Objectives

When you complete this chapter, you will be able to:

- Enhance form usability with JavaScript
- Customize browser-based HTML validation
- Implement custom validation to check for errors and display error messages

Using JavaScript with Forms

- Validation
 - checking that form information provided by users conforms to data rules
- `form object`
 - Represents a form in an HTML document
 - Used to access form and its data

Using JavaScript with Forms (cont'd.)

PROPERTY	DESCRIPTION
autocomplete	Enables autocompletion of previously saved form data by a browser when set to <code>true</code>
elements	Returns a collection of a form's elements
length	Returns an integer representing the number of elements in the form
novalidate	Disables browser-based validation by a browser when set to <code>false</code>

Table 6-1 Properties of `form` objects

EVENT	DESCRIPTION
submit	Fires when a form's submit button is clicked

Table 6-2 Event of `form` objects

METHOD	DESCRIPTION
<code>checkValidity()</code>	Initiates browser-based validation of form controls, returning <code>true</code> if all controls are valid
<code>submit()</code>	Submits a form without the use of a submit button

Table 6-3 Methods of `form` objects

Using JavaScript with Forms (cont'd.)

- Common elements for collecting form data:
 - input
 - select
 - option
 - textarea
 - button

Working with Input Fields (cont'd.)

PROPERTY	DESCRIPTION
autofocus	Returns the value <code>true</code> if the HTML <code>autofocus</code> attribute is set, indicating that the element should receive the focus when the form is loaded
placeholder	Returns the value of the <code>placeholder</code> attribute, which contains text to be displayed when a field has no value
required	Returns the value <code>true</code> if the HTML <code>required</code> attribute is set, indicating that the control must contain a value before the form can be submitted
validationMessage	Sets or returns the text of the message to be displayed to the user after a failed <code>submit</code> event if the field's <code>validity</code> value is <code>false</code>
validity	Returns the value <code>true</code> if the control value passes browser-based validation rules
value	Sets or returns the value of the control
willValidate	Returns the value <code>true</code> if constraint validation is enabled for the control

Table 6-4 Properties of elements within forms

Working with Input Fields (cont'd.)

METHOD	DESCRIPTION
<code>checkValidity()</code>	Runs constraint validation against a form element; returns a value of <code>true</code> if the value is valid, and <code>false</code> if the value is invalid
<code>setCustomValidity ("message")</code>	Sets the string <code>message</code> as the text to be displayed to users if the control value is found to be invalid; passing a nonempty string sets the control's validity to <code>false</code> , and passing an empty string as the parameter sets the control's validity to <code>true</code>

Table 6-5 Methods of elements within forms

EVENT	TRIGGERED WHEN
<code>blur</code>	The focus leaves the element, meaning that the element is initially selected or contains the insertion point, and then another element is selected or contains the insertion point (usually by clicking another element or pressing Tab to move to another element)
<code>change</code>	The focus leaves the element, and the value or selected state of the current element has changed
<code>focus</code>	The element receives the focus, meaning the element is selected or the insertion point moves into the current element (usually by clicking the current element or pressing Tab to move to the current element)
<code>formchange</code>	Another control in the form fires a <code>change</code> event
<code>forminput</code>	Another control in the form fires an <code>input</code> event
<code>input</code>	The value or selected state of the current element changes
<code>invalid</code>	A control's value is found to be invalid during constraint validation

Table 6-6 Events of elements within forms

Referencing Forms and Form Elements

- Can use `getElementsByName()` method:
`getElementsByName("form") [0]`
- Document object includes a `forms []` array
 - Contains all forms on a web page
- `form` object has an `elements []` array

Referencing Forms and Form Elements (cont'd.)

- `elements [] array`
 - Contains objects representing each control in a form
- Reference form index number in the `forms [] array`
 - Followed by the appropriate element index number from the `elements [] array`

Improving Form Usability

- Before validation
 - Can reduce amount of validation necessary

Designing Forms to Collect More Accurate Content

- Replace input boxes with other fields that present limited choices

Designing Forms to Collect More Accurate Content (cont'd.)

NAME	CODE TO CREATE	USE TO DISPLAY
Option buttons	<code><input type="radio" /></code>	A small set of options at once, from which a user can select one
Check boxes	<code><input type="checkbox" /></code>	One or more yes/no choices
Selection lists	<code><select> <option>value1</option> <option>value2</option> ... </select></code>	A truncated list of options that's fully displayed when a user interacts with it
Sliders	<code><input type="range" /></code>	A bar with an indicator that users can drag to increase or decrease a value; used with <code>min</code> and <code>max</code> attributes to specify bottom and top of range (supported in IE10+ and all other modern browsers)
Data lists	<code><input type="text" list="listname" /> <datalist id="listname"> <option value="value1" /> ... </datalist></code>	A text box that suggests values from the <code>datalist</code> element as user types, but enables a user to enter a value not in the list

Table 6-7 Selected form elements for providing limited choices

Designing Forms to Collect More Accurate Content (cont'd.)

Payment

Card Type

Card #

Expiration

CVV

Figure 6-2 Sample fieldset containing `input` elements

Payment

Visa Master Card Discover American Express

Card #

Expiration

CVV

Figure 6-3 Sample fieldset updated with option buttons and selection lists

Programming Forms to Increase Content Accuracy

- Assistive functions
 - Reduce likelihood of user errors
 - Prevent users from entering erroneous data
- Removing default values from selection lists
 - Can set default value for selection list in HTML
 - Only to one of the options
 - JavaScript can set `selectedIndex` property to -1
 - Corresponds to no selection

Programming Forms to Increase Content Accuracy (cont'd.)

PROPERTY	DESCRIPTION
<code>length</code>	Returns the number of <code>option</code> elements nested within the <code>select</code> element
<code>multiple</code>	Sets or returns a Boolean value that determines whether multiple options can be selected in the selection list
<code>options</code>	Returns a collection of the elements nested within the <code>select</code> element
<code>selectedIndex</code>	Returns a number representing the element number in the <code>options</code> collection of the first option selected in a selection list; returns <code>-1</code> if no option is selected
<code>size</code>	Sets or returns the number of options to be displayed at once
<code>type</code>	Returns the type of selection list, which is either <code>select-one</code> if the <code>select</code> element does not include the <code>multiple</code> attribute, or <code>select-multiple</code> if the <code>select</code> element does include the <code>multiple</code> attribute

Table 6-8 `select` element properties

Programming Forms to Increase Content Accuracy (cont'd.)

- Dynamically Updating Selection List Values
 - Can add or remove `option` elements from a `select` element using node methods
 - Can change list options based on selection in another field

Programming Forms to Increase Content Accuracy (cont'd.)

PROPERTY	DESCRIPTION
defaultSelected	Returns a Boolean value that determines whether the <code>option</code> element representing the currently selected item includes the <code>selected</code> attribute
index	Returns a number representing the element number within the <code>options</code> collection
label	Sets or returns alternate text to be displayed for the option in the selection list
selected	Sets or returns a Boolean value that determines whether an option is selected
text	Sets or returns the text displayed for the option in the selection list
value	Sets or returns the text that is assigned to the <code>option</code> element's <code>value</code> attribute; this value is submitted to the server

Table 6-9 Properties of `option` elements

Programming Forms to Increase Content Accuracy (cont'd.)

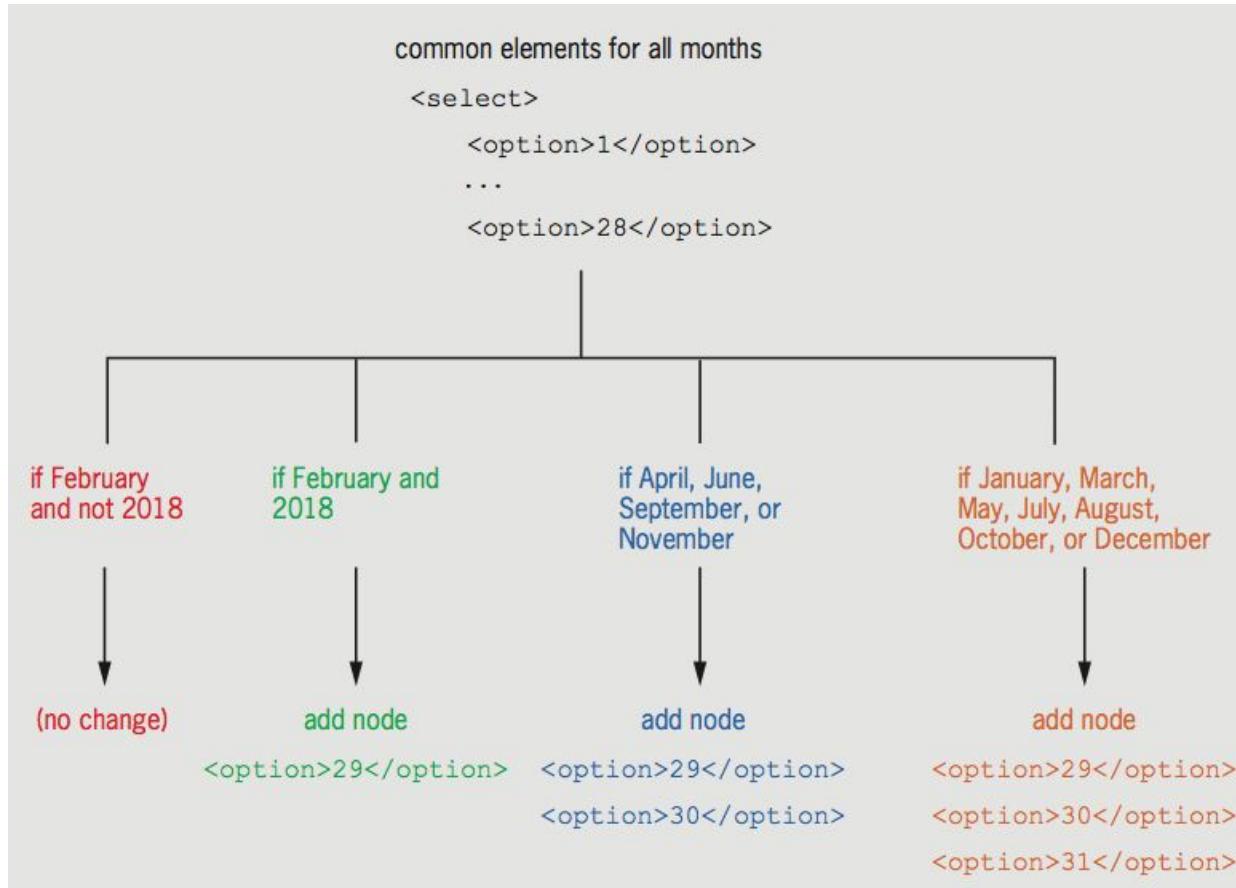


Figure 6-5 Diagram of function for dynamically updating selection list values

Programming Forms to Increase Content Accuracy (cont'd.)

- Adding Placeholder Text for Older Browsers
 - placeholder attribute of input and textarea elements
 - Supported by modern browsers
 - Can recreate behavior with JavaScript for older browsers:
 - Add placeholder text when page finishes loading
 - Remove placeholder text when user selects field
 - Add back placeholder text if user makes no entry

Programming Forms to Increase Content Accuracy (cont'd.)

- Automatically updating an associated field based on a user entry
 - Multiple elements may be associated
 - Example: check box to indicate `textarea` entry
 - Can automatically change value of one field in response to change in other field

Programming Forms to Increase Content Accuracy (cont'd.)

- Transferring duplicate field values
 - Can copy data from one field to another based on user indicating they should have the same value
 - Example: Shipping Address and Billing Address

Programming Forms to Increase Content Accuracy (cont'd.)

The screenshot shows a web form with two address sections: **Billing Address** and **Delivery Address**. The **Billing Address** section contains fields for First Name (Maria), Last Name (Santiago), Street Address (1 Main St), City (Las Cruces), State (NM), Zip (88001), and Phone (575-555-2000). The **Delivery Address** section contains fields for First Name (Maria), Last Name (Santiago), Street Address (1 Main St), City (Las Cruces), State (NM), Zip (88001), and Phone (575-555-2000). A green callout box points to the **Delivery Address** section, stating: "checking 'same as billing address' box copies Billing Address control values to Delivery Address controls". Another green callout box points to the same row of fields in the **Delivery Address** section, stating: "values copied from corresponding fields in Billing Address section".

Billing Address	
First Name	Maria
Last Name	Santiago
Street Address	1 Main St
City	Las Cruces
State	NM
Zip	88001
Phone	575-555-2000

Delivery Address	
<input checked="" type="checkbox"/> same as billing address	
First Name	Maria
Last Name	Santiago
Street Address	1 Main St
City	Las Cruces
State	NM
Zip	88001
Phone	575-555-2000

Figure 6-10 Billing Address entries copied to Delivery Address section

Customizing Browser-Based Validation

- Modern browsers can perform some validation
 - Known as browser-based validation, native validation, or HTML5 validation

Customizing Browser-Based Validation (cont'd.)

- Specifying browser-based validation parameters
 - Use attributes listed in Table 6-12

ATTRIBUTE	DESCRIPTION	USE WITH
formnovalidate	Toggles off validation of the form when added to the <input> tag for a submit button	input elements with a type value of submit
max	Specifies the control's maximum numerical value	input elements with a type value of number
maxlength	Specifies the control's maximum number of characters	textarea elements, or input elements displayed as a text box
min	Specifies the control's minimum numerical value	input elements with a type value of number
novalidate	Toggles off validation of the form when added to the opening <form> tag	the form element
pattern	Specifies a pattern that a control's value must match, expressed as a regular expression	textarea elements, or input elements displayed as a text box
required	Indicates that the control must have a value	input, select, or textarea elements
step	Specifies the increment that a numerical value must adhere to	input elements with a type value of number

Table 6-12 HTML attributes to set browser-based validation parameters

Customizing Browser-Based Validation (cont'd.)

- Specifying browser-based validation parameters
 - Additional validation linked to `input type` values

VALUE	CONTENT DESCRIPTION	SAMPLE CONTENT
<code>color</code>	# followed by a 6-digit hexadecimal color value, with any letters in lower case	# ffcc00
<code>date</code> , <code>datetime</code> , <code>week</code> , <code>month</code> , <code>time</code> , <code>datetime-local</code>	Relevant components of coordinated universal time (UTC), a standardized date/time format	2019-10-14T12:00:00.001-04:00
<code>email</code>	An email address	president@whitehouse.gov
<code>number</code>	A numerical value	73.54

Table 6-13 Values for `type` attribute that trigger browser-based validation

Customizing Browser-Based Validation (cont'd.)

- Customizing browser-based validation feedback
 - Modern browsers display feedback in similar ways, with variation
 - Displayed after `submit` event triggered
 - Invalid controls highlighted
 - Bubble displayed next to first control

Customizing Browser-Based Validation (cont'd.)

- Customizing browser-based validation feedback (cont'd.)
 - Customizable through constraint validation API
 - All properties of `validity` object must have value of `false` for element to be valid

Customizing Browser-Based Validation (cont'd.)

PROPERTY	RETURNS <code>true</code> IF
<code>customError</code>	A custom error message has been set with <code>setCustomValidity()</code>
<code>patternMismatch</code>	The control value does not match the value of the <code>pattern</code> attribute
<code>rangeOverflow</code>	The control value is greater than the value of the <code>max</code> attribute
<code>rangeUnderflow</code>	The control value is less than the value of the <code>min</code> attribute
<code>stepMismatch</code>	The control value does not conform to the value of the <code>step</code> attribute
<code>tooLong</code>	The length of the control value is greater than the value of the <code>maxlength</code> attribute
<code>typeMismatch</code>	The control value does not conform to the rules for the <code>type</code> attribute value
<code>valueMissing</code>	The control value is empty but the control has a <code>required</code> attribute set
<code>valid</code>	None of the preceding properties are <code>true</code>

Table 6-14 `validity` properties

Customizing Browser-Based Validation (cont'd.)

- Customizing browser-based validation feedback (cont'd.)
 - `checkValidity()` and `setCustomValidity()` methods
 - CSS `:invalid` and `:valid` pseudo-classes
 - Use to change properties of form elements based on validity status

Customizing Browser-Based Validation (cont'd.)

- Customizing browser-based validation feedback
(cont'd.)

JavaScript

```
var fname = document.getElementById("firstName") ;  
  
if (fname.valueMissing) {  
    setCustomValidity("Please fill out this field.");  
}  
  
css  
#firstName:invalid {  
    background: rgb(255,233,233);  
}
```

Customizing Browser-Based Validation (cont'd.)

- Customizing browser-based validation feedback
(cont'd.)

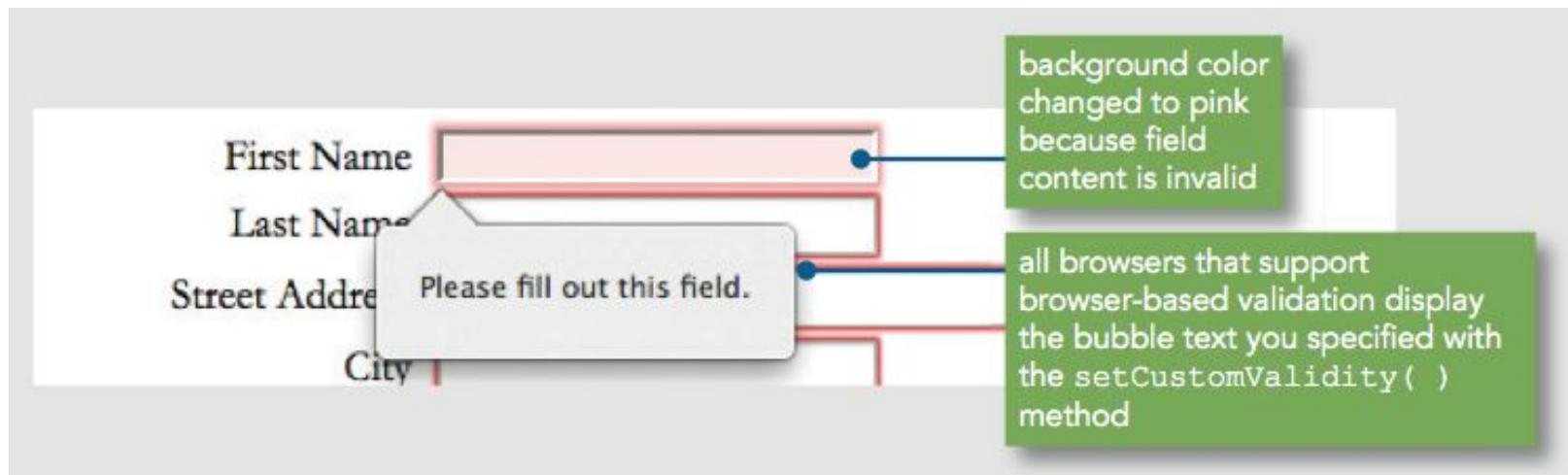


Figure 6-13 Customized browser-based validation

Customizing Browser-Based Validation (cont'd.)

- Customizing browser-based validation feedback (cont'd.)
 - Bubble appearance varies among browsers
 - Cannot set multiple validation messages for a single field at once
 - Can disable browser-based validation using the `preventDefault()` method and the `invalid` event
 - If disabled, must program custom validation

Programming Custom Validation

- Common validation functions:
 - Checking that required fields contain entries
 - Checking values dependent on other fields
 - Checking for appropriate content type

Validating Submitted Data

- submit event fires when a form is submitted
 - Often when submit button selected
 - Data usually validated when submit event fires
 - preventDefault() method disables default behavior of an event when it fires
 - Not supported in IE8, so set returnValue to false instead

Validating Required Fields with Custom Functions

- Retrieve values of required fields, then check if any is empty

```
try {  
    if (element.value === "") {  
        throw "message";  
    }  
}  
  
catch(message) {  
    // code to display message and highlight error  
    formValidity = false;  
}
```

Validating Required Fields with Custom Functions (cont'd.)

- Checking for empty text input fields
 - Check value property for a value

```
if (document.getElementById("firstName").value === "") {  
  
    // code to run if the field is blank  
  
}
```

- Use loop statement to check each field in a group

Validating Required Fields with Custom Functions (cont'd.)

- Checking for selection lists with no values
 - Check value of selectedIndex property
 - If no option is selected, value is -1

```
if (document.getElementById("state").selectedIndex === -1) {  
  
    // code to run if the field is blank  
  
}
```

Validating Required Fields with Custom Functions (cont'd.)

- Checking for option button sets with no selection
 - Check value of checked property
 - Use And (`&&`) operators to check if no option button is selected

```
var buttons = document.getElementsByName("Color");  
  
if (!buttons[0].checked && !buttons[1].checked && ←  
    !buttons[2].checked) {  
  
    // code to run if no button is selected  
  
}
```

Validating Dependent Fields with Custom Functions

- Sometimes need to test logic specific to a form
- Validating based on the state of a check box
 - Access same checked property used with option button
- Validating based on text input box contents
 - Can use nested if statements to account for possibilities when entry in one text box requires entry in another

Validating Dependent Fields with Custom Functions (cont'd.)

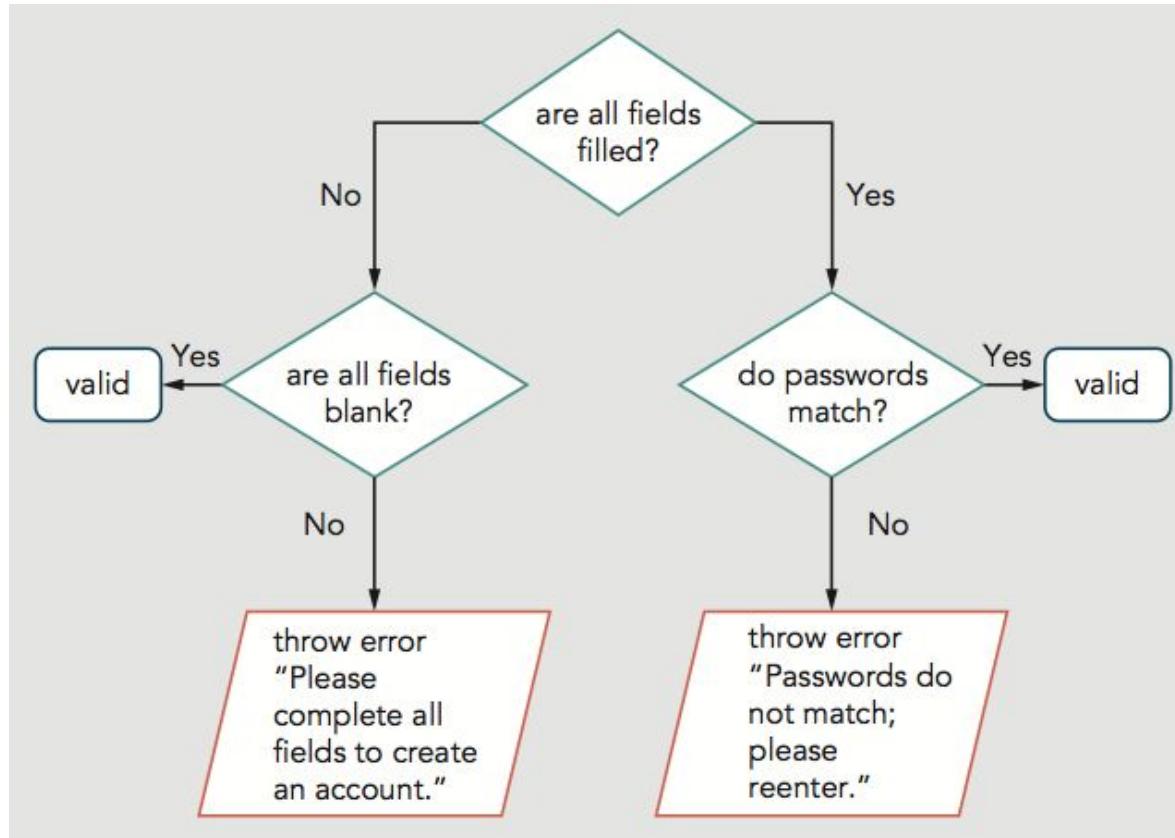


Figure 6-21 Flowchart diagram of `validateCreateAccount()` function

Validating Content Type with Custom Functions

- Can check whether numeric fields contain numbers
 - Use `isNaN()` function
 - returns true if value is not a number
- Character patterns like zip codes require regular expressions (Chapter 8)

```
isNaN(document.getElementById("subtotal").value)
```

Summary

- Validation checks that information conforms to rules
- Assistive functions reduce likelihood of user errors
- Browser-based validation is built into modern browsers
 - Customizable through Constraint Validation API
- `preventDefault()` method blocks action normally associated with an event

Summary (cont'd.)

- To validate required text input fields
 - Retrieve the values of the required fields
 - Check if the value of any of them is an empty string
- To validate required selection lists
 - Retrieve the `selectedIndex` value
 - Check whether it's equal to -1

Summary (cont'd.)

- To check if an option button is selected, access the value of its checked property.
- To check if none of the option buttons in a set are selected, create a conditional statement using And (& &) operators
- In some cases, you need to create validation functions to test logic specific to your form

Javascript

Chapter 7
Using Object-Oriented JavaScript

Objectives

When you complete this chapter, you will be able to:

- Explain basic concepts related to object-oriented programming
- Use the Date, Number, and Math objects
- Define your own custom JavaScript objects

Introduction to Object-Oriented Programming

- Object-oriented programming
 - Allows reuse of code without having to copy or recreate it

Reusing Software Objects

- Object-oriented programming (OOP)
 - Creating reusable software objects
 - Easily incorporated into multiple programs
- Object
 - Programming code and data treated as an individual unit or component
 - Also called a component
- Data
 - Information contained within variables or other types of storage structures

Reusing Software Objects (cont'd.)

- Objects range from simple controls to entire programs
- Popular object-oriented programming languages
 - C++, Java, Visual Basic

Reusing Software Objects (cont'd.)

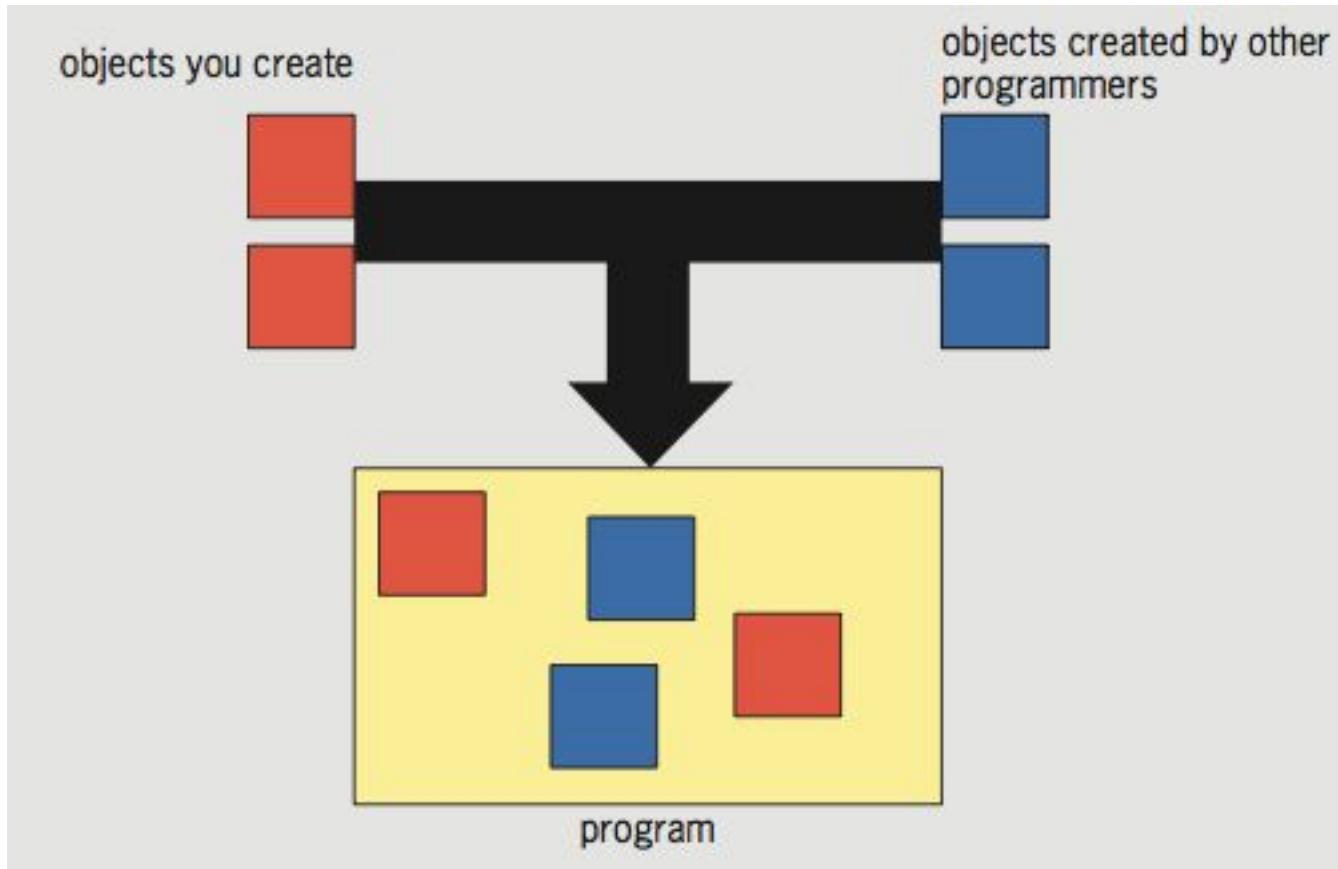


Figure 7-1 Programming with objects

What Is Encapsulation?

- Encapsulated objects
 - Code and data contained within the object itself
- Encapsulation places code inside a “black box”
- Interface
 - Elements required for program to communicate with an object
- Principle of information hiding
 - Any methods and properties other programmers do not need to access should be hidden

What Is Encapsulation? (cont'd.)

- Advantages of encapsulation
 - Reduces code complexity
 - Prevents accidental bugs and stealing of code
- Programming object and its interface
 - Compare to a handheld calculator

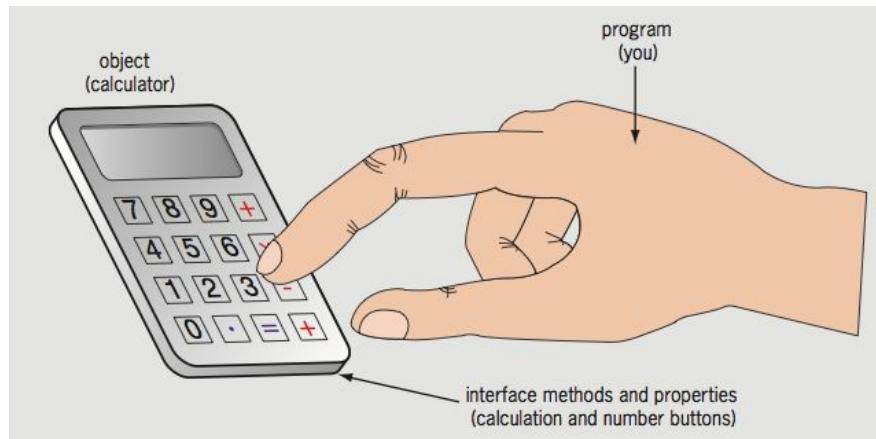


Figure 7-2 Calculator interface

What Is Encapsulation? (cont'd.)

- Document object is encapsulated (black box)
 - getElementById() method
 - Part of the interface JavaScript uses to communicate with the Document object
- Microsoft Word: example of an object and its interface

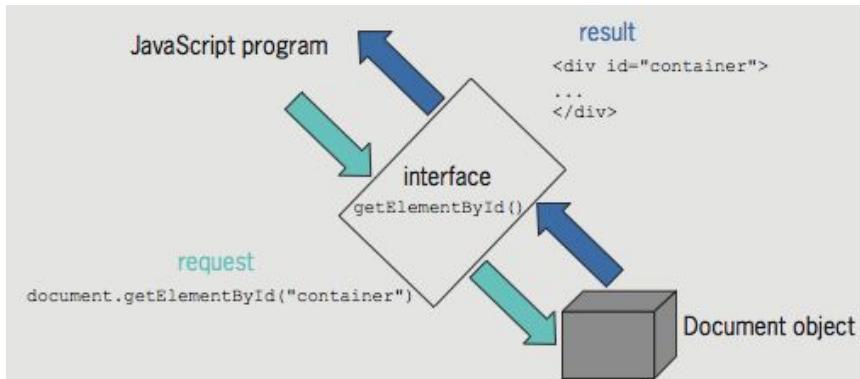


Figure 7-3 Using the interface for the Document object

Understanding Classes

- Classes
 - Grouping of code, methods, attributes, etc., making up an object
- Instance
 - Object created from an existing class
- Instantiate: create an object from an existing class
- Instance of an object inherits its methods and properties from a class
- Objects in the browser object model
 - Part of the web browser
 - No need to instantiate them to use them

Using Built-In JavaScript Classes

CLASS	DESCRIPTION
Arguments	Retrieves and manipulates arguments within a function
Array	Creates new array objects
Boolean	Creates new Boolean objects
Date	Retrieves and manipulates dates and times
Error	Returns run-time error information
Function	Creates new function objects
Global	Stores global variables and contains various built-in JavaScript functions
JSON	Manipulates objects formatted in JavaScript Object Notation (JSON); available in ECMAScript 5 and later
Math	Contains methods and properties for performing mathematical calculations
Number	Contains methods and properties for manipulating numbers
Object	Represents the base class for all built-in JavaScript classes; contains several of the built-in JavaScript functions
RegExp	Contains methods and properties for finding and replacing characters in text strings
String	Contains methods and properties for manipulating text strings

Table 7-1 Built-in JavaScript classes

Using Built-In JavaScript Classes (cont'd.)

- Instantiating an object
 - Some of the built-in JavaScript objects used directly in code
 - Some objects require programmer to instantiate a new object
 - Example: Math object's PI (π) property in a script

```
// calculate the area of a circle based on its radius
function calcCircleArea() {
    var r = document.getElementById("radius").value;
    var area = Math.PI * Math.pow(r, 2); // area is pi times ←
    radius squared
    return area;
}
```

Using Built-In JavaScript Classes (cont'd.)

- Instantiating an object (cont'd.)
 - Can instantiate **Array** object using array literal
 - Example: `var deptHeads = [];`
 - Can instantiate empty generic object using object literal
 - Example: `var accountsPayable = {};`
 - Generic object literal uses curly braces around value
 - Can't use object literal for **Date** object
 - Must use constructor
 - Example: `var today = new Date();`

Using Built-In JavaScript Classes (cont'd.)

- Performing garbage collection
 - Garbage collection
 - Cleaning up, or reclaiming, memory reserved by a program
 - Declaring a variable or instantiating a new object
 - Reserves memory for the variable or object
 - JavaScript knows when a program no longer needs a variable or object
 - Automatically cleans up the memory

Using the Date, Number, and Math Classes

- Three of most commonly used JavaScript classes:
 - Date, Number, and Math

Manipulating the Date and Time with the Date Class

- Date class
 - Methods and properties for manipulating the date and time
 - Allows use of a specific date or time element in JavaScript programs

CONSTRUCTOR	DESCRIPTION
<code>Date()</code>	Creates a Date object that contains the current date and time provided by the device
<code>Date(milliseconds)</code>	Creates a Date object based on the number of milliseconds that have elapsed since midnight, January 1, 1970
<code>Date(date_string)</code>	Creates a Date object based on a string containing a date value
<code>Date(year, month[, day, hours, minutes, seconds, milliseconds])</code>	Creates a Date object with the date and time set according to the passed arguments; the <code>year</code> and <code>month</code> arguments are required

Table 7-2 Date class constructors

Manipulating the Date and Time with the Date Class (cont'd.)

- Example:
 - `var today = new Date();`
 - Month and year date representation in a `Date` object
 - Stored using numbers matching actual date and year
- Days of the week and months of the year
 - Stored using numeric representations
 - Starting with zero: similar to an array
- Example:
 - `var independenceDay = new Date(1776, 6, 4);`

Manipulating the Date and Time with the Date Class (cont'd.)

- After creating a new Date object
 - Manipulate date and time in the variable using the Date class methods
- Date and time in a Date object
 - Not updated over time like a clock
 - Date object contains the static (unchanging) date and time
 - Set at the moment the JavaScript code instantiates the object

Manipulating the Date and Time with the Date Class (cont'd.)

METHOD	DESCRIPTION
<code>getDate()</code>	Returns the date of a Date object
<code>getDay()</code>	Returns the day of a Date object
<code>getFullYear()</code>	Returns the year of a Date object in four-digit format
<code>getHours()</code>	Returns the hour of a Date object
<code>getMilliseconds()</code>	Returns the milliseconds of a Date object
<code>getMinutes()</code>	Returns the minutes of a Date object
<code>getMonth()</code>	Returns the month of a Date object
<code>getSeconds()</code>	Returns the seconds of a Date object
<code>getTime()</code>	Returns the time of a Date object
<code>now()</code>	Returns the current time as the number of milliseconds that have elapsed since midnight, January 1, 1970 (ECMAScript 5 and later only)

Table 7-3 Commonly used methods of the `Date` class (*continues*)

Manipulating the Date and Time with the Date Class (cont'd.)

METHOD	DESCRIPTION
<code> setDate (<i>date</i>)</code>	Sets the date (1–31) of a Date object
<code> setFullYear (<i>year</i>[, <i>month</i>, <i>day</i>])</code>	Sets the four-digit year of a Date object; optionally allows you to set the month and the day
<code> setHours (<i>hours</i>[, <i>minutes</i>, <i>seconds</i>, <i>milliseconds</i>])</code>	Sets the hours (0–23) of a Date object; optionally allows you to set the minutes (0–59), seconds (0–59), and milliseconds (0–999)
<code> setMilliseconds (<i>milliseconds</i>)</code>	Sets the milliseconds (0–999) of a Date object
<code> setMinutes (<i>minutes</i>[, <i>seconds</i>, <i>milliseconds</i>])</code>	Sets the minutes (0–59) of a Date object; optionally allows you to set seconds (0–59) and milliseconds (0–999)
<code> setMonth (<i>month</i>[, <i>date</i>])</code>	Sets the month (0–11) of a Date object; optionally allows you to set the date (1–31)
<code> setSeconds (<i>seconds</i>[, <i>milliseconds</i>])</code>	Sets the seconds (0–59) of a Date object; optionally allows you to set milliseconds (0–999)
<code> setTime ()</code>	Sets the time as the number of milliseconds that have elapsed since midnight, January 1, 1970
<code> toLocaleString ()</code>	Converts a Date object to a string, set to the current time zone
<code> toString ()</code>	Converts a Date object to a string
<code> valueOf ()</code>	Converts a Date object to a millisecond format

Table 7-3 Commonly used methods of the Date class

Manipulating the Date and Time with the Date Class (cont'd.)

- Each portion of a Date object can be retrieved and modified using the Date object methods

- Examples:

```
var curDate = new Date();  
curDate.getDate();
```

- Displaying the full text for days and months
 - Use a conditional statement to check the value returned by the getDay() or getMonth() method
 - Example:
 - if/else construct to print the full text for the day of the week returned by the getDay() method

Manipulating the Date and Time with the Date Class (cont'd.)

```
var today = new Date();

var curDay = today.getDay();

var weekday;

if (curDay === 0) {

    weekday = "Sunday";

} else if (curDay === 1) {

    weekday = "Monday";

} else if (curDay === 2) {

    weekday = "Tuesday";

} else if (curDay === 3) {

    weekday = "Wednesday";

} else if (curDay === 4) {

    weekday = "Thursday";
```

Manipulating the Date and Time with the Date Class (cont'd.)

- Example: include an array named months
 - 12 elements assigned full text names of the months

```
var today = new Date();  
  
var months = ["January", "February", "March", ←  
              "April", "May", "June", ←  
              "July", "August", "September", ←  
              "October", "November", "December"];  
  
var curMonth = months[today.getMonth()];
```

Manipulating Numbers with the Number Class

- Number class
 - Methods for manipulating numbers and properties containing static values
 - Representing some numeric limitations in the JavaScript language
 - Can append the name of any Number class method or property
 - To the name of an existing variable containing a numeric value

Manipulating Numbers with the Number Class (cont'd.)

- Using Number class methods

METHOD	DESCRIPTION
<code>toExponential(<i>decimals</i>)</code>	Converts a number to a string in exponential notation using the number of decimal places specified by <i>decimals</i>
<code>toFixed(<i>decimals</i>)</code>	Converts a number to a string using the number of decimal places specified by <i>decimals</i>
<code>toLocaleString()</code>	Converts a number to a string that is formatted with local numeric formatting style
<code>toPrecision(<i>decimals</i>)</code>	Converts a number to a string with the number of decimal places specified by <i>decimals</i> , in either exponential notation or in fixed notation
<code>toString(<i>base</i>)</code>	Converts a number to a string using the number system specified by <i>base</i>
<code>valueOf()</code>	Returns the numeric value of a Number object

Table 7-4 Number class methods

Manipulating Numbers with the Number Class (cont'd.)

- Using Number class methods (cont'd.)
 - Primary reason for using any of the “to” methods
 - To convert a number to a string value with a specific number of decimal places
 - toFixed() method
 - Most useful Number class method
 - toLocaleString() method
 - Converts a number to a string formatted with local numeric formatting conventions

Manipulating Numbers with the Number Class (cont'd.)

- Accessing Number class properties

PROPERTY	DESCRIPTION
MAX_VALUE	The largest positive number that can be used in JavaScript
MIN_VALUE	The smallest positive number that can be used in JavaScript
NaN	The value NaN, which stands for "not a number"
NEGATIVE_INFINITY	The value of negative infinity
POSITIVE_INFINITY	The value of positive infinity

Table 7-5 Number class properties

Performing Math Functions with the Math Class

- Math **class**
 - Methods and properties for mathematical calculations
- Cannot instantiate a Math object using a statement such as: `var mathCalc = new Math();`
 - Use the Math object and one of its methods or properties directly in the code
- Example:

```
var curNumber = 144;
```

```
var squareRoot = Math.sqrt(curNumber); // returns 12
```

Performing Math Functions with the Math Class (cont'd.)

METHOD	RETURNS
<code>abs(x)</code>	The absolute value of x
<code>acos(x)</code>	The arc cosine of x
<code>asin(x)</code>	The arc sine of x
<code>atan(x)</code>	The arc tangent of x
<code>atan2(x, y)</code>	The angle from the x-axis of the point represented by x, y
<code>ceil(x)</code>	The value of x rounded to the next highest integer
<code>cos(x)</code>	The cosine of x
<code>exp(x)</code>	The exponent of x
<code>floor(x)</code>	The value of x rounded to the next lowest integer
<code>log(x)</code>	The natural logarithm of x
<code>max(x, y)</code>	The larger of x or y
<code>min(x, y)</code>	The smaller of x or y
<code>pow(x, y)</code>	The value of x raised to the y power
<code>random()</code>	A random number
<code>round(x)</code>	The value of x rounded to the nearest integer
<code>sin(x)</code>	The sine of x
<code>sqrt(x)</code>	The square root of x
<code>tan(x)</code>	The tangent of x

Table 7-6 Math class methods

Performing Math Functions with the Math Class (cont'd.)

PROPERTY	DESCRIPTION
E	Euler's constant e , which is the base of a natural logarithm; this value is approximately 2.7182818284590452354
LN10	The natural logarithm of 10, which is approximately 2.302585092994046
LN2	The natural logarithm of 2, which is approximately 0.6931471805599453
LOG10E	The base-10 logarithm of e , the base of the natural logarithms; this value is approximately 0.4342944819032518
LOG2E	The base-2 logarithm of e , the base of the natural logarithms; this value is approximately 1.4426950408889634
PI	A constant representing the ratio of the circumference of a circle to its diameter, which is approximately 3.1415926535897932
SQRT1_2	The square root of 1/2, which is approximately 0.7071067811865476
SQRT2	The square root of 2, which is approximately 1.4142135623730951

Table 7-7 Math class properties

Performing Math Functions with the Math Class (cont'd.)

- Example:
 - Use the `PI` property to calculate the area of a circle based on its radius
 - Code uses the `pow()` method to raise the radius value to second power, and the `round()` method to round the value returned to the nearest whole number

```
var radius = 25;  
  
var area = Math.PI * Math.pow(radius, 2);  
  
var roundedArea = Math.round(area); // returns 1963
```

Defining Custom JavaScript Objects

- JavaScript: not a true object-oriented programming language
 - Cannot create classes in JavaScript
 - Instead, called an object-based language
- Can define custom objects
 - Not encapsulated
 - Useful to replicate the same functionality an unknown number of times in a script

Declaring Basic Custom Objects

- Use the Object object
 - `var objectName = new Object();`
 - `var objectName = {};`
- Can assign properties to the object
 - Append property name to the object name with a period

Declaring Basic Custom Objects (cont'd.)

- Add properties using dot syntax
 - Object name followed by dot followed by property name
 - Example:

```
InventoryList.inventoryDate = new Date(2017, 11, 31);
```

Declaring Basic Custom Objects (cont'd.)

- Can assign values to the properties of an object when object first instantiated
- Example:

```
var PerformanceTickets = {  
    customerName: "Claudia Salomon",  
    performanceName: "Swan Lake",  
    ticketQuantity: 2,  
    performanceDate: new Date(2017, 6, 18, 20)  
};
```

Declaring Sub-Objects

- Value of a property can be another object
 - called a sub-object
 - Example—order object with address sub-object:

```
var order = {  
  
    orderNumber: "F5987",  
  
    address: {  
  
        street: "1 Main St",  
  
        city: "Farmington",  
  
        state: "NY",  
  
        zip: "14425"  
    }  
}
```

Referring to Object Properties as Associative Arrays

- Associative array
 - An array whose elements are referred to with an alphanumeric key instead of an index number
- Can also use associative array syntax to refer to the properties of an object
- With associative arrays
 - Can dynamically build property names at runtime

Referring to Object Properties as Associative Arrays (cont'd.)

- Can use associative array syntax to refer to the properties of an object
- Example:

```
var stopLightColors = {  
    stop: "red",  
    caution: "yellow",  
    go: "green"  
};  
stopLightColors["caution"];
```

Referring to Object Properties as Associative Arrays (cont'd.)

- Can easily reference property names that contain numbers
 - Example:

```
var order = {  
  
    item1: "KJ2435J",  
  
    price1: 23.95,  
  
    item2: "AW23454",  
  
    price2: 44.99,  
  
    item3: "2346J3B",  
  
    price3: 9.95  
};
```

Referring to Object Properties as Associative Arrays (cont'd.)

- Can easily reference property names that contain numbers (cont'd.)
 - To create order summary:

```
for (var i = 1; i < 4; i++) {  
  
    document.getElementById("itemList").innerHTML += <  
        "p class='item'>" + order["item" + i] + "</p>";  
  
    document.getElementById("itemList").innerHTML += <  
        "p class='price'>" + order["price" + i] + "</p>";  
  
};
```

Referring to Object Properties as Associative Arrays (cont'd.)

- Can also write generic code to add new object properties that incorporate numbers
 - Example—adding items to shopping cart:

```
totalItems += 1; // increment counter of items in order

currentItem = document.getElementById("itemName").innerHTML;

currentPrice = document.getElementById("itemPrice").innerHTML;

newItemPropertyName = "item" + totalItems; // "item4"

newPricePropertyName = "price" + totalItems; // "price4"

order.newItemPropertyName = currentItem; // order.item4 = (name)

order.newPricePropertyName = currentPrice;

// order.price4 = (price);
```

Creating Methods

- Object method simply a function with a name within the object
- Two ways to add method to object
 - Provide code for method in object
 - Reference external function

Creating Methods (cont'd.)

- Specify method name with anonymous function as value
 - Example:

```
var order = {  
    items: {},  
    generateInvoice: function() {  
        // function statements  
    }  
};
```

Creating Methods (cont'd.)

- Specify method name with existing function as value
 - Example:

```
function processOrder() {  
    // function statements  
}  
  
var order = {  
    items: {},  
    generateInvoice: processOrder  
};
```

- Reference to existing function cannot have parentheses

Enumerating custom object properties

- Custom objects can contain dozens of properties
- To execute the same statement or command block for all the properties within a custom object
 - Use the `for/in` statement
 - Looping statement similar to the `for` statement
- Syntax

```
for (variable in object) {  
  
    statement(s);  
  
}
```

Enumerating custom object properties (cont'd.)

- `for/in` statement enumerates, or assigns an index to, each property in an object
- Typical use:
 - validate properties within an object

Enumerating custom object properties (cont'd.)

- Example—checking for empty values:

```
var item = {  
  
    itemNumber: "KJ2435J",  
  
    itemPrice: 23.95,  
  
    itemInStock: true,  
  
    itemShipDate: new Date(2017, 6, 18),  
  
};  
  
for (prop in order) {  
  
    if (order[prop] === "") {  
  
        order.generateErrorMessage();  
  
    }  
}
```

Deleting Properties

- Use the `delete` operator
- Syntax

`delete object.property`

- Example:

`delete order.itemInStock;`

Defining Constructor Functions

- Constructor function
 - Used as the basis for a custom object
 - Also known as object definition
- JavaScript objects
 - Inherit all the variables and statements of the constructor function on which they are based
- All JavaScript functions
 - Can serve as a constructor

Defining Constructor Functions (cont'd.)

- Example:
 - Define a function that can serve as a constructor function

```
function Order(number, order, payment, ship) {  
  
    this.customerNumber = number;  
  
    this.orderDate = order;  
  
    this.paymentMethod = payment;  
  
    this.shippingDate = ship;  
}
```

Adding Methods to a Constructor Function

- Can create a function to use as an object method
 - Refer to object properties with `this` reference
 - Example:

```
function displayOrderInfo() {  
  
    var summaryDiv = document.getElementById("summarySection");  
  
    summaryDiv.innerHTML += "<p>Customer: " +  
        this.customerNumber + "</p>";  
  
    summaryDiv.innerHTML += "<p>Order Date: " +  
        this.orderDate.toLocaleString() + "</p>";  
  
    summaryDiv.innerHTML += "<p>Payment: " +  
        this.paymentMethod + "</p>";  
  
    summaryDiv.innerHTML += "<p>Ship Date: " +  
        this.shipDate + "</p>";  
}
```

Using the prototype Property

- After instantiating a new object
 - Can assign additional object properties
 - Use a period
- New property only available to that specific object
- prototype property
 - Built-in property that specifies the constructor from which an object was instantiated
 - When used with the name of the constructor function
 - Any new properties you create will also be available to the constructor function

Using the prototype Property (cont'd.)

- Object definitions can use the `prototype` property to extend other object definitions
 - Can create a new object based on an existing object

Summary

- Object-oriented programming (or OOP)
 - The creation of reusable software objects
- Reusable software objects
 - Called components
- Object
 - Programming code and data treated as an individual unit or component
- Objects are encapsulated
- Interface represents elements required for a source program to communicate with an object

Summary (cont'd.)

- Principle of information hiding
- Code, methods, attributes, and other information that make up an object
 - Organized using classes
- Instance
 - Object created from an existing class
- An object inherits the characteristics of the class on which it is based
- Date class contains methods and properties for manipulating the date and time

Summary (cont'd.)

- Number class contains methods for manipulating numbers and properties
- Math class contains methods and properties for performing mathematical calculations
- Can define custom object
 - object literal
- Can create template for custom objects
 - constructor function
- this keyword refers to object that called function
- prototype property specifies object's constructor

Javascript

Chapter 8
Manipulating Data in Strings and Arrays

Objectives

When you complete this chapter, you will be able to:

- Manipulate strings with properties and methods of the `String` object
- Create regular expressions and use them to validate user input
- Manipulate arrays with properties and methods of the `Array` object
- Convert between strings and arrays, and between strings and JSON

Manipulating Strings

- String
 - Text contained within double or single quotation marks
 - Literal values or assigned to a variable
 - Begin and end with same type of quotation mark
- Example:

```
document.getElementById("mainHeading").innerHTML = "24-Hour  
Forecast";  
  
var highSurfAdvisory = "Watch out for high waves and strong  
rip currents.";
```

Manipulating Strings (cont'd.)

- Parsing
 - Extracting characters or substrings from a larger string
- Use `String` class to parse text strings in scripts
 - Represents all literal strings and string variables in JavaScript
 - Contains methods for manipulating text strings

Formatting Strings

- Using special characters
 - For basic types: use escape sequences

```
var mainHead = 'Today\'s Forecast';
```

- For other special characters: use Unicode
 - Standardized set of characters from many of the world's languages

```
copyrightInfo = "<p>&#169; 2006-2017</p>";
```

```
// numeric character ref.
```

```
copyrightInfo = "<p>&copy; 2006-2017</p>";
```

```
// character entity
```

Formatting Strings (cont'd.)

- Using special characters (cont'd.)
 - `fromCharCode()` method
 - Constructs a text string from Unicode character codes
 - Syntax:
`String.fromCharCode(char1, char2, ...)`

- Examples:
`String.fromCharCode(74, 97, 118, 97, 83, 99, 114, 105, 112, 116)`

```
copyrightInfo = String.fromCharCode(169) + " 2017";
```

Formatting Strings (cont'd.)

- Changing case
 - `toLowerCase()` and `toUpperCase()` methods
 - Examples:

```
var agency = "noaa";  
  
agencyName.innerHTML = agency.toUpperCase();  
  
// browser displays "NOAA" but value of agency is still "noaa"
```

```
var agency = "noaa";  
  
agency = agency.toUpperCase();  
  
// value of agency is "NOAA"
```

```
agencyName.innerHTML = agency;  
  
// browser displays "NOAA"
```

Counting Characters in a String

- `length` property
 - Returns the number of characters in a string
 - Example:

```
var country = "Kingdom of Morocco";
```

```
var stringLength = country.length;
```

```
// value of stringLength is 18
```

Finding and Extracting Characters and Substrings

METHOD	DESCRIPTION
<code>charAt(index)</code>	Returns the character at the specified position in a text string; returns an empty string if the specified position is greater than the length of the string
<code>charCodeAt(index)</code>	Returns the Unicode character code at the specified position in a text string; returns <code>Nan</code> if the specified position is greater than the length of the string
<code>indexOf(text[, index])</code>	Performs a case-sensitive search and returns the position number in a string of the first character in the <code>text</code> argument; if the <code>index</code> argument is included, then the <code>indexOf()</code> method starts searching at that position within the string; returns <code>-1</code> if the character or string is not found
<code>lastIndexOf(text[, index])</code>	Performs a case-sensitive search and returns the position number in a string of the last instance of the first character in the <code>text</code> argument; if the <code>index</code> argument is included, then the <code>lastIndexOf()</code> method starts searching at that position within the string; returns <code>-1</code> if the character or string is not found

Table 8-1 Search and extraction methods of the `String` class (*continues*)

Finding and Extracting Characters and Substrings (cont'd.)

METHOD	DESCRIPTION
<code>match(pattern)</code>	Performs a case-sensitive search and returns an array containing the results that match the <i>pattern</i> argument; returns <i>null</i> if the text is not found
<code>search(pattern)</code>	Performs a case-sensitive search and returns the position number in a string of the first instance of the first character in the <i>pattern</i> argument; returns –1 if the character or string is not found
<code>slice(starting index [, ending index])</code>	Extracts text from a string, starting with the position number in the string of the <i>starting index</i> argument and ending with the character immediately before the position number of the <i>ending index</i> argument; allows negative argument values
<code>substring(starting index [, ending index])</code>	Extracts text from a string, starting with the position number in the string of the <i>starting index</i> argument and ending with the character immediately before the position number of the <i>ending index</i> argument; does not allow negative argument values

Table 8-1 Search and extraction methods of the `String` class

Finding and Extracting Characters and Substrings (cont'd.)

string	"	s	m	i	t	h	@	e	x	a	m	p	l	e	.	c	o	m	"
index values	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16		
reverse index values	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1		
method																			
result																			
charAt(5)																			
"@"																			
charCodeAt(5)																			
40																			
indexOf("com")																			
14																			
lastIndexOf("e")																			
12																			
search("@")																			
5																			
slice(-11,-4)																			
"example"																			
substring(6,13)																			
"example"																			

Figure 8-5 Example uses of String class methods

Finding and Extracting Characters and Substrings (cont'd.)

- Two types of string search methods
 - Those that return a numeric position in a text string
 - Character position in text string begins with a value of zero
 - Can pass a second optional argument specifying the position in the string to start searching to the `indexOf()` method
 - Example: `search()` method

```
var email = "president@whitehouse.gov";
```

```
var atPosition = email.search("@"); // returns 9
```

Finding and Extracting Characters and Substrings (cont'd.)

- Two types of string search methods (cont'd.)
 - Those that return a numeric position in a text string (cont'd.)
 - Example: `indexOf()` method

```
var email = "president@whitehouse.gov";
```

```
var atIndex = email.indexOf("@", 10); // returns -1
```

Finding and Extracting Characters and Substrings (cont'd.)

- Two types of string search methods (cont'd.)
 - Those that return a character or substring

- `substring()` or `slice()` method

```
var email = "president@whitehouse.gov";
```

```
var nameEnd = email.search("@");
```

```
// value of nameEnd is 9
```

```
var nameText = email.substring(0, nameEnd);
```

```
// value of nameText is "president"
```

Finding and Extracting Characters and Substrings (cont'd.)

- Extracting characters from the middle or end of a string
 - Use the `search()`, `indexOf()`, `lastIndexOf()` methods
 - `lastIndexOf()` method returns position of the last occurrence of one string in another string
 - Example:

```
var email = "president@whitehouse.gov";  
  
var startDomainID = email.lastIndexOf(".") ;  
  
// startDomainID value is 20  
  
var domainID = email.substring(startDomainID + 1);  
  
// domainID value is "gov"
```

Finding and Extracting Characters and Substrings (cont'd.)

- `slice()` method allows negative argument values for the index arguments
 - Specifying a negative value for the starting index
 - `slice()` method starts at the end of the text string
 - Specifying a negative value for the ending index
 - Number of characters the `slice()` method extracts also starts at the end of the text string
- `slice()` method does not return the character represented by the ending index
 - Returns the character immediately before the ending index

Finding and Extracting Characters and Substrings (cont'd.)

```
var email = "president@whitehouse.gov";  
  
var nameText = email.slice(0, 9);  
  
// nameText value is "president"  
  
var domain = email.slice(-14, -4);  
  
// domain value is "whitehouse"
```

Replacing Characters and Substrings

- `replace()` method
 - Creates a new string with the first instance of a specified pattern replaced with the value of the `text` argument
 - **Syntax:** `string.replace(pattern, text)`
 - **Example:**

```
var email = "president@whitehouse.gov";  
  
var newEmail = email.replace("president", "vice.president");  
  
// value of newEmail is "vice.president@whitehouse.gov"
```

Combining Characters and Substrings

- Combining strings
 - Use the concatenation operator (+) and compound assignment operator (+=)
 - Use the `concat()` method
 - Creates a new string by combining strings passed as arguments
 - Syntax: `string.concat(value1, value2, ...)`
 - To combine text strings
 - Easier to use the concatenation operator and the compound assignment operator

Combining Characters and Substrings (cont'd.)

```
var name = "Theodor Seuss Geisel";  
  
var penName = "Dr. Seuss";  
  
var bio = penName.concat(" was the pen name of ", name);  
  
// value of bio is  
  
// "Dr. Seuss was the pen name of Theodor Seuss Geisel"  
var name = "Theodor Seuss Geisel";  
  
var penName = "Dr. Seuss";  
  
var bio = penName + " was the pen name of " + name;  
  
// value of bio is  
  
// "Dr. Seuss was the pen name of Theodor Seuss Geisel"
```

Comparing Strings

- Comparison operator (==) can be used with strings
 - Compare individual characters according to their Unicode position
- `localeCompare()` method
 - Compares strings according to the particular sort order of a language or country
 - Performs a case-sensitive comparison of two strings

Working with Regular Expressions

- Regular expressions
 - Patterns used for matching and manipulating strings according to specified rules
 - With scripting languages, most commonly used for validating submitted form data

Defining Regular Expressions in JavaScript

- Regular expressions
 - Must begin and end with forward slashes
 - Example: `var urlProtocol = /https/;`
- Approaches to creating regular expressions
 - Use regular expressions with several String class methods
 - Pass pattern directly to a method
 - Use the `RegExp()` constructor
 - Contains methods and properties for working with regular expressions in JavaScript

Defining Regular Expressions in JavaScript (cont'd.)

- Approaches to creating regular expressions (cont'd.)
 - Syntax for creating a regular expression with the `RegExp()` constructor

```
var regExpName = new RegExp("pattern[, attributes]");
```

- Example:

```
var urlProtocol = new RegExp("https");

var url = "http://www.cengagebrain.com";

var searchResult = url.search(urlProtocol);

// returns -1
```

Using Regular Expression Methods

- RegExp object
 - Includes two methods
 - `test()` and `exec()`
- `test()` method: returns a value of true
 - If a string contains text that matches a regular expression
 - Otherwise returns false value
 - Syntax: `var pattern = test(string);`
- Real power of regular expressions
 - Comes from the patterns written

Writing Regular Expression Patterns

- Hardest part of working with regular expressions
 - Writing the patterns and rules
 - Example:

```
var emailPattern = /^[_a-zA-Z0-9\-\]+(\.\[_a-zA-Z0-9\-\]+\+)*@[_a-zA-Z0-9\-\]+(\.\[_a-zA-Z0-9\-\]+\+)*(\.\[a-z]\{2,6\})$/;

var email = "president@whitehouse.gov";

var result;

if (emailPattern.test(email)) {

    result = true;

} else {

    result = false;
```

Writing Regular Expression Patterns (cont'd.)

- Regular expression patterns consist of literal characters and metacharacters
 - Metacharacters: special characters that define the pattern matching rules in a regular expression

Writing Regular Expression Patterns (cont'd.)

METACHARACTER	DESCRIPTION
.	Matches any single character
\	Identifies the next character as a literal value
^	Matches characters at the beginning of a string
\$	Matches characters at the end of a string
()	Specifies required characters to include in a pattern match
[]	Specifies alternate characters allowed in a pattern match
[^]	Specifies characters to exclude in a pattern match
-	Identifies a possible range of characters to match
	Specifies alternate sets of characters to include in a pattern match

Table 8-2 JavaScript regular expression metacharacters

Writing Regular Expression Patterns (cont'd.)

- Matching any character
 - Period (.)
 - Matches any single character in a pattern
 - Specifies that the pattern must contain a value where the period located
- Matching characters at the beginning or end of a string
 - ^ metacharacter
 - Matches characters at the beginning of a string
 - \$ metacharacter
 - Matches characters at the end of a string

Writing Regular Expression Patterns (cont'd.)

- Matching characters at the beginning or end of a String (cont'd.)
 - Anchor
 - Pattern that matches the beginning or end of a line
 - Specifying an anchor at the beginning of a line
 - Pattern must begin with the ^ metacharacter
- Matching special characters
 - Precede the character with a backslash

Writing Regular Expression Patterns (cont'd.)

- Specifying quantity
 - Quantifiers
 - Metacharacters that specify the quantity of a match

QUANTIFIER	DESCRIPTION
?	Specifies that the preceding character is optional
+	Specifies that one or more of the preceding characters must match
*	Specifies that zero or more of the preceding characters can match
{ <i>n</i> }	Specifies that the preceding character repeat exactly <i>n</i> times
{ <i>n</i> ,}	Specifies that the preceding character repeat at least <i>n</i> times
{ <i>n</i> 1, <i>n</i> 2}	Specifies that the preceding character repeat at least <i>n</i> 1 times but no more than <i>n</i> 2 times

Table 8-3 JavaScript regular expression quantifiers

Writing Regular Expression Patterns (cont'd.)

- Specifying subexpressions
 - Subexpression or subpattern
 - Characters contained in a set of parentheses within a regular expression
 - Allows determination of the format and quantities of the enclosed characters as a group

Writing Regular Expression Patterns (cont'd.)

- Defining character classes
 - Character classes
 - Used in regular expressions to treat multiple characters as a single item
 - Created by enclosing the characters that make up the class with bracket [] metacharacters
 - Use a hyphen metacharacter (-) to specify a range of values in a character class
 - To specify optional characters to exclude in a pattern match
 - Include the ^ metacharacter immediately before the characters in a character class

Writing Regular Expression Patterns (cont'd.)

- Defining character classes (cont'd.)
 - Regular expressions include special escape characters in character classes
 - To represent different types of data

EXPRESSION	DESCRIPTION
\w	Alphanumeric characters
\W	Any character that is not an alphanumeric character
\d	Numeric characters
\D	Nonnumeric characters
\s	White space characters
\S	All printable characters
\b	Backspace character

Table 8-4 JavaScript character class expressions

Writing Regular Expression Patterns (cont'd.)

- Defining character classes (cont'd.)
 - Matching multiple pattern choices
 - Allow a string to contain an alternate set of substrings
 - Separate the strings in a regular expression pattern with the | metacharacter

Setting Regular Expression Properties

PROPERTY	FLAG	DESCRIPTION
global	g	Determines whether to search for all possible matches within a string
ignoreCase	i	Determines whether to ignore letter case when executing a regular expression
lastIndex		Stores the index of the first character from the last match (no flag)
multiline	m	Determines whether to search across multiple lines of text
source		Contains the regular expression pattern (no flag)

Table 8-5 Properties of the `RegExp` object

Setting Regular Expression Properties (cont'd.)

- Options for setting the values of these properties
 - Assign a value of true or false to the property
 - By creating a regular expression with the `RegExp()` constructor
 - Use the flags when assigning a regular expression to a variable without using the `RegExp()` constructor

Manipulating Arrays

- Use the `Array` class `length` property and methods
- Creating an array
 - Instantiates an object from the `Array` class

Manipulating Arrays (cont'd.)

METHOD	DESCRIPTION
<code>array1.concat(array2 [, array3, ...])</code>	Combines arrays
<code>pop()</code>	Removes the last element from the end of an array
<code>push(value1[, value2, ...])</code>	Adds one or more elements to the end of an array, where <code>value1</code> , <code>value2</code> , etc., are the values to add
<code>reverse()</code>	Reverses the order of the elements in an array
<code>shift()</code>	Removes and returns the first element from the beginning of an array
<code>slice(start, end)</code>	Copies a portion of an array to another array, where <code>start</code> is the array index number at which to begin extracting elements, and <code>end</code> is an integer value that indicates the number of elements to return from the array
<code>sort()</code>	Sorts an array alphabetically
<code>splice(start, elements_ to_delete[, value1, value2, ...])</code>	Adds or removes elements within an array, where <code>start</code> indicates the index number within the array where elements should be added or removed; <code>elements_to_delete</code> is an integer value that indicates the number of elements to remove from the array, starting with the element indicated by the <code>start</code> argument; and <code>value1</code> , <code>value2</code> , etc., represent the values to add
<code>unshift(value1[, value2, ...])</code>	Adds one or more elements to the beginning of an array, where <code>value1</code> , <code>value2</code> , etc., are the values to add

Table 8-6 Methods of the `Array` class

Finding and Extracting Elements and Values

- Primary method for finding a value in an array
 - Use a looping statement to iterate through the array until a particular value found
- Extract elements and values from an array
 - Use the `slice()` method to return (copy) a portion of an array and assign it to another array
- Syntax for the `slice()` method

`array_name.slice(start, end);`

Finding and Extracting Elements and Values (cont'd.)

```
var largestStates = ["Alaska", "Texas", "California",  
                     "Montana", "New Mexico", "Arizona", "Nevada",  
                     "Colorado", "Oregon", "Wyoming"];  
  
var fiveLargestStates = largestStates.slice(0, 5);  
  
for (var i = 0; i < fiveLargestStates.length; i++) {  
  
    var newItem = document.createElement("p");
```

```
Alaska  
Texas  
California  
Montana  
New Mexico
```

Figure 8-11 List of states extracted using the `slice()` method

Manipulating Elements

- Adding and removing elements to and from the beginning of an array
 - `shift()` method removes and returns the first element from the beginning of an array
 - `unshift()` method adds one or more elements to the beginning of an array

Manipulating Elements (cont'd.)

```
var colors = ["mauve", "periwinkle", "silver", "cherry",  
             "lemon"] ;  
  
colors.shift(); // colors value now  
// ["periwinkle", "silver", "cherry", "lemon"]  
  
colors.unshift("yellow-orange", "violet") ;  
  
// colors value now ["yellow-orange", "violet",  
// "mauve", "periwinkle", "silver", "cherry", "lemon"]
```

Manipulating Elements

- Adding and removing elements to and from the end of an array
 - Use array's `length` property to determine the next available index

```
var colors = ["mauve", "periwinkle", "silver", "cherry"];  
  
colors[colors.length] = "lemon";  
  
// colors value now ["mauve", "periwinkle", "silver",  
// "cherry", "lemon"]
```

Manipulating Elements (cont'd.)

- Adding and removing elements to and from the end of an array (cont'd.)
 - `pop()` method removes the last element from the end of an array
 - `push()` method adds one or more elements to the end of an array

```
var colors = ["mauve", "periwinkle", "silver", "cherry"];  
  
colors.pop();  
  
// colors value now ["mauve", "periwinkle", "silver"]  
  
colors.push("yellow-orange", "violet");  
  
// colors value now ["mauve", "periwinkle", "silver",  
// "yellow-orange", "violet"]
```

Manipulating Elements (cont'd.)

- Adding and removing elements within an array
 - Use the `splice()` method
 - Also renames the indexes in the array
 - To add an element, include 0 as second argument

```
var hospitalDepts = ["Anesthesia", "Molecular Biology",  
                      "Neurology", "Pediatrics"];  
  
hospitalDepts.splice(3, 0, "Ophthalmology");  
  
// value now ["Anesthesia", "Molecular Biology",  
// "Neurology", "Ophthalmology", "Pediatrics"]
```

Manipulating Elements (cont'd.)

- Adding and removing elements within an array (cont'd.)
 - Use the `splice()` method (cont'd.)
 - To delete elements, omit third argument
 - Indexes renumbered just like when elements added

```
var hospitalDepts = ["Anesthesia", "Molecular Biology",  
"Neurology", "Pediatrics"];  
  
hospitalDepts.splice(1, 2);  
  
// value now ["Anesthesia", "Pediatrics"]
```

Sorting and Combining Arrays

- Sorting arrays
 - Sort elements of an array alphabetically
 - Use the `sort()` method

```
var scientificFishNames = ["Quadratus taiwanae",  
  "Macquaria australasica", "Jordania zonope",  
  "Abudefdup sparoides", "Dactylopterus volitans",  
  "Wattsia mossambica", "Bagrus urostigma"];  
  
scientificFishNames.sort();  
  
// scientificFishNames value now  
// ["Abudefdup sparoides", "Bagrus urostigma",  
//  "Dactylopterus volitans", "Jordania zonope",  
//  "Macquaria australasica", "Quadratus taiwanae",  
//  "Wattsia mossambica"]
```

Sorting and Combining Arrays (cont'd.)

- Sorting arrays (cont'd.)
 - `reverse()` method
 - Transposes, or reverses, the order of the elements in an array

```
scientificFishNames.reverse();  
  
// scientificFishNames value now  
  
// ["Wattsia mossambica", "Quadratus taiwanae",  
// "Macquaria australasica", "Jordania zonope",  
// "Dactylopterus volitans", "Bagrus urostigma",  
// "Abudedefduf sparoides"]
```

Sorting and Combining Arrays (cont'd.)

- Combining arrays
 - Use the `concat()` method
 - Syntax

```
array1.concat(array2, array3, ...);
```

Sorting and Combining Arrays (cont'd.)

```
var Provinces = ["Newfoundland and Labrador",  
                 "Prince Edward Island", "Nova Scotia",  
                 "New Brunswick", "Quebec", "Ontario",  
                 "Manitoba", "Saskatchewan", "Alberta",  
                 "British Columbia"];  
  
var Territories = ["Nunavut", "Northwest Territories",  
                   "Yukon"];  
  
var Canada = [];  
  
Canada = Provinces.concat(Territories);  
  
// value of Canada now ["Newfoundland and Labrador",  
// "Prince Edward Island", "Nova Scotia",  
// "New Brunswick", "Quebec", "Ontario",  
// "Manitoba", "Saskatchewan", "Alberta",  
// "British Columbia"];
```

Converting Between Data Types

- Common task to convert strings and arrays to different data types
 - strings to arrays
 - arrays to strings
 - objects to strings
 - strings to objects

Converting Between Strings and Arrays

- `split()` method of the `String` class
 - Splits a string into an indexed array

- **Syntax**

`array = string.split(separator[, limit]);`

- To split individual characters in a string into an array
 - Pass an empty string ("") as the separator argument

Converting Between Strings and Arrays (cont'd.)

```
var OPEC = "Algeria, Angola, Ecuador, Iran, Iraq, Kuwait,←  
Libya, Nigeria, Qatar, Saudi Arabia,←  
United Arab Emirates, Venezuela";  
  
// The value of OPEC is a string  
  
var opecArray = OPEC.split(", ");  
  
// The value of opecArray is the following array:  
  
// ["Algeria", "Angola", "Ecuador", "Iran", "Iraq",  
// "Kuwait", "Libya", "Nigeria", "Qatar", "Saudi Arabia",  
// "United Arab Emirates", "Venezuela"]
```

Converting Between Strings and Arrays (cont'd.)

- `join()` method of the `Array` class
 - Combines array elements into a string, separated by a comma or specified characters
- Syntax

```
array.join(["separator"]);
```

- To prevent elements from being separated by any characters in the new string
 - Pass an empty string ("") as the separator argument

Converting Between Strings and Arrays (cont'd.)

```
var OPEC = ["Algeria", "Angola", "Ecuador", "Iran",  
           "Iraq", "Kuwait", "Libya", "Nigeria", "Qatar",  
           "Saudi Arabia", "United Arab Emirates", "Venezuela"];  
  
// value of OPEC is an array  
  
var opecString = OPEC.join();  
  
// value of opecString is the following string:  
  
// "Algeria, Angola, Ecuador, Iran, Iraq, Kuwait, Libya,  
// Nigeria, Qatar, Saudi Arabia, United Arab Emirates,  
// Venezuela"
```

Converting Between Strings and Arrays (cont'd.)

- `join()` method does not include a separator argument
 - Previous example OPEC nations automatically separated by commas
 - Can include a separator argument of `";"`

```
var OPEC = ["Algeria", "Angola", "Ecuador", "Iran",  
           "Iraq", "Kuwait", "Libya", "Nigeria", "Qatar",  
           "Saudi Arabia", "United Arab Emirates", "Venezuela"];  
  
// value of OPEC is an array  
  
var opecString = OPEC.join(";" );  
  
// value of opecString is the following string:  
  
// "Algeria;Angola;Ecuador;Iran;Iraq;Kuwait;Libya;  
// Nigeria;Qatar;Saudi Arabia;United Arab Emirates"
```

Converting Between Strings and Arrays (cont'd.)

- Can also use the `toString()` and `toLocaleString()` method
 - Convert an array to a string
 - `array.toString();`
 - `array.toLocaleString();`

Converting Between Strings and JSON

- JavaScript Object Notation (JSON)
 - Represents a JavaScript object as a string
 - Exchanges data between application and server
- JSON object
 - Supported in modern browsers, including IE8

METHOD	DESCRIPTION
<code>parse()</code>	Converts a string value to an object
<code>stringify()</code>	Converts an object to a string value

Table 8-7 Methods of the `JSON` object

Converting Between Strings and JSON (cont'd.)

- Converting an Object to a String
 - `stringify()` method
 - `string = JSON.stringify(value [, replacer [, space]])`
 - `string` is name of variable that will contain string
 - `value` represents JavaScript object to be converted

Converting Between Strings and JSON (cont'd.)

- Converting an Object to a String (cont'd.)

```
var newUser = {  
    fName: "Tony",  
    lName: "Chu"  
},  
  
newUserString = JSON.stringify(newUser);  
  
// value of newUserString is  
  
// '{ "fName": "Tony", "lName": "Chu" }'
```

Converting Between Strings and JSON (cont'd.)

- Converting a String to an Object
 - `parse()` method
 - `object` `=` `JSON.parse(string [, function]);`
 - `object` **is** name of variable that will contain object
 - `string` represents JSON string to be converted

Converting Between Strings and JSON (cont'd.)

- Converting a String to an Object (cont'd.)
 - JSON string definition:

```
var newUser = '{"fName": "Tony", "lName": "Chu"}';
```

- String because enclosed in quotes
 - To convert string to JavaScript object:

```
var newUserObject = JSON.parse(newUser);  
  
// value of newUserObject is  
  
// {  
//     fName: "Tony",  
//     lName: "Chu"  
// };
```

Summary

- Parsing
 - Act of extracting characters or substrings from a larger string
- All literal strings and string variables in JavaScript
 - Represented by the `String` class
- `fromCharCode()` method of the `String` class
 - Constructs a text string from Unicode character codes
- `toLowerCase()` and `toUpperCase()` methods
 - Change the case of letters in a string

Summary (cont'd.)

- String class
 - length property
 - **Methods:** replace(), concat(), localeCompare()
- Regular expressions
 - Patterns used for matching and manipulating strings according to specified rules
- RegExp object
 - Contains methods and properties for working with regular expressions in JavaScript

Summary (cont'd.)

- Use the `Array` class methods and `length` property to manipulate arrays in scripts
 - Methods: `slice()`, `shift()` and `unshift()`, `pop()` and `push()`, `splice()`, `sort()`, `reverse()`, `concat()`, and `join()`
- `split()` method of the `String` class
 - Splits a string into an indexed array
- `join()` method of the `Array` class
 - Combines array elements into a string

Summary (cont'd.)

- Use the `JSON class` methods to convert between string values and object values
- `stringify()` method of the `JSON class`
 - Converts JavaScript object to JSON string
- `parse()` method of the `JSON class`
 - Converts JSON string to JavaScript object