

JavaScript

by Example

Second Edition

EXAMPLE

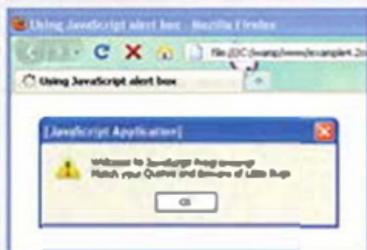
using the alert() method

```
<html>
<head>
<title>Using JavaScript alert box</title>
<script type="text/javascript">
1   var message1="Match your Quotes and ";
2   var message2="Beware of Little Bugs ";
3   alert("Welcome to JavaScript Programming!\n" +
4     message1 + message2);
</script>
</html>
```

EXPLANATION

- 1 The JavaScript program starts here with the `<script>` tag.
- 2 Two variables, `message1` and `message2` are assigned text strings.
- 3 The `alert()` method contains a string of text. Buried in the string is a backslash `escape sequence`. There are a number of these sequences available in JavaScript (see Table 3.1 on page 32). The `\n` causes a line break in a string. The reason for using the `\n` in escape sequence is to display the text on two lines.
- 4 The `alert()` method receives two arguments: the text string and the variables. The `+` sign concatenates the variables together. As shown in the screenshot, the program v

RESULT



Ellie Quigley

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com

Library of Congress Cataloging-in-Publication Data

Quigley, Ellie.

JavaScript by example / Ellie Quigley.—2nd ed.

Includes index.

ISBN 978-0-13-705489-3 (pbk. : alk. paper)

1. JavaScript (Computer program language) I. Title.

QA76.73.J39Q54 2010

005.13'3—dc22

2010020402

Copyright © 2011 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax: (617) 671-3447

ISBN-13: 978-0-13-705489-3

ISBN-10: 0-13-705489-0

Text printed in the United States on recycled paper at Edwards Brothers in Ann Arbor, Michigan.

First printing, October 2010

Editor-in-Chief

Mark L. Taub

Managing Editor

John Fuller

Full-Service

Production Manager

Julie B. Nahil

Production Editor

Dmitri Korzh
Techne Group

Copy Editor

Teresa Horton

Indexer

Potomac Indexing, LLC

Proofreader

Beth Roberts

Editorial Assistant

Kim Boedigheimer

Cover Designer

Anne Jones

Composition

Techne Group

Contents

Preface xv

1	Introduction to JavaScript	1
1.1	What JavaScript Is	1
1.2	What JavaScript Is Not	2
1.3	What JavaScript Is Used For	3
1.4	JavaScript and Its Place in a Web Page	4
1.4.1	Analysis of the Diagram	4
1.5	What Is Ajax?	5
1.6	What JavaScript Looks Like	7
1.7	JavaScript and Its Role in Web Development	8
1.7.1	The Three Layers	8
1.8	JavaScript and Events	10
1.9	Standardizing JavaScript and the W3C	12
1.9.1	JavaScript Objects	13
1.9.2	The Document Object Model	13
1.10	What Browser?	15
1.10.1	Versions of JavaScript	16
1.10.2	Does Your Browser Follow the Standard?	18
1.10.3	Is JavaScript Enabled on Your Browser?	18
1.11	Where to Put JavaScript	20
1.11.1	JavaScript from External Files	22
1.12	Validating Your Markup	24
1.12.1	The W3C Validation Tool	24
1.12.2	The Validome Validation Tool	25
1.13	What You Should Know	26

- 2 Script Setup 29**
 - 2.1 The HTML Document and JavaScript 29
 - 2.1.1 Script Execution 30
 - 2.2 Syntactical Details 33
 - 2.2.1 Case Sensitivity 33
 - 2.2.2 Free Form and Reserved Words 33
 - 2.2.3 Statements and Semicolons 34
 - 2.2.4 Comments 35
 - 2.2.5 The `<script>` Tag 35
 - 2.3 Generating HTML and Printing Output 37
 - 2.3.1 Strings and String Concatenation 37
 - 2.3.2 The `write()` and `writeln()` Methods 38
 - 2.4 About Debugging 40
 - 2.4.1 Types of Errors 40
 - 2.5 Debugging Tools 41
 - 2.5.1 Firefox 41
 - 2.5.2 Debugging in Internet Explorer 8 44
 - 2.5.3 The `JavaScript:` URL Protocol 46
 - 2.6 JavaScript and Old or Disabled Browsers 47
 - 2.6.1 Hiding JavaScript from Old Browsers 47
 - 2.7 What You Should Know 50

- 3 The Building Blocks: Data Types, Literals, and Variables 53**
 - 3.1 Data Types 53
 - 3.1.1 Primitive Data Types 53
 - 3.1.2 Composite Data Types 59
 - 3.2 Variables 59
 - 3.2.1 Valid Names 60
 - 3.2.2 Declaring and Initializing Variables 60
 - 3.2.3 Dynamically or Loosely Typed Language 62
 - 3.2.4 Scope of Variables 66
 - 3.2.5 Concatenation and Variables 66
 - 3.3 Constants 67
 - 3.4 Bugs to Watch For 69
 - 3.5 What You Should Know 70

- 4 Dialog Boxes 73**
 - 4.1 Interacting with the User 73
 - 4.1.1 The `alert()` Method 73
 - 4.1.2 The `prompt()` Method 76
 - 4.1.3 The `confirm()` Method 78
 - 4.2 What You Should Know 80

5 Operators 83

- 5.1 About JavaScript Operators and Expressions 83
 - 5.1.1 Assignment 84
 - 5.1.2 Precedence and Associativity 84
- 5.2 Types of Operators 88
 - 5.2.1 Arithmetic Operators 88
 - 5.2.2 Shortcut Assignment Operators 90
 - 5.2.3 Autoincrement and Autodecrement Operators 91
 - 5.2.4 Concatenation Operator 94
 - 5.2.5 Comparison Operators 95
 - 5.2.6 Logical Operators 101
 - 5.2.7 The Conditional Operator 108
 - 5.2.8 Bitwise Operators 109
- 5.3 Number, String, or Boolean? Data Type Conversion 112
 - 5.3.1 The *parseInt()* Function 114
 - 5.3.2 The *parseFloat()* Function 116
 - 5.3.3 The *eval()* Function 118
- 5.4 Special Operators 119
- 5.5 What You Should Know 120

6 Under Certain Conditions 123

- 6.1 Control Structures, Blocks, and Compound Statements 123
- 6.2 Conditionals 123
 - 6.2.1 *if/else* 124
 - 6.2.2 *if/else if* 127
 - 6.2.3 *switch* 128
- 6.3 Loops 131
 - 6.3.1 The *while* Loop 131
 - 6.3.2 The *do/while* Loop 133
 - 6.3.3 The *for* Loop 134
 - 6.3.4 The *for/in* Loop 135
 - 6.3.5 Loop Control with *break* and *continue* 136
 - 6.3.6 Nested Loops and Labels 137
- 6.4 What You Should Know 140

7 Functions 143

- 7.1 What Is a Function? 143
 - 7.1.1 Function Declaration and Invocation 144
 - 7.1.2 Return Values 153
 - 7.1.3 Anonymous Functions as Variables 156
 - 7.1.4 Closures 158
 - 7.1.5 Recursion 161
 - 7.1.6 Functions Are Objects 166

- 7.2 Debugging Techniques 166
 - 7.2.1 Function Syntax 166
 - 7.2.2 Exception Handling with *try/catch* and *throw* 168
- 7.3 What You Should Know 172

8 Objects 175

- 8.1 What Are Objects? 175
 - 8.1.1 Objects and the Dot Syntax 176
 - 8.1.2 Creating an Object with a Constructor 177
 - 8.1.3 Properties of the Object 178
 - 8.1.4 Methods of the Object 180
- 8.2 Classes and User-Defined Functions 182
 - 8.2.1 What Is a Class? 182
 - 8.2.2 What Is *this*? 182
 - 8.2.3 Inline Functions as Methods 185
- 8.3 Object Literals 187
- 8.4 Manipulating Objects 191
 - 8.4.1 The *with* Keyword 191
 - 8.4.2 The *for/in* Loop 194
- 8.5 Extending Objects with Prototypes 196
 - 8.5.1 Adding Properties with the Prototype Property 198
 - 8.5.2 The Prototype Lookup Chain 199
 - 8.5.3 Adding Methods with Prototype 202
 - 8.5.4 Properties and Methods of All Objects 204
 - 8.5.5 Creating Subclasses and Inheritance 207
- 8.6 What You Should Know 210

9 JavaScript Core Objects 213

- 9.1 What Are Core Objects? 213
- 9.2 Array Objects 213
 - 9.2.1 Declaring and Populating Arrays 214
 - 9.2.2 *Array* Object Properties 219
 - 9.2.3 Associative Arrays 221
 - 9.2.4 Nested Arrays 223
- 9.3 Array Methods 227
- 9.4 The *Date* Object 234
 - 9.4.1 Using the *Date* Object Methods 235
 - 9.4.2 Manipulating the Date and Time 238
 - 9.4.3 Customizing the *Date* Object with the *prototype* Property 240
- 9.5 The *Math* Object 241
 - 9.5.1 Rounding Up and Rounding Down 244
 - 9.5.2 Generating Random Numbers 245

9.5.3	Wrapper Objects (String, Number, Function, Boolean)	246
9.5.4	The <i>String</i> Object	247
9.5.5	The <i>Number</i> Object	259
9.5.6	The <i>Boolean</i> Object	263
9.5.7	The <i>Function</i> Object	264
9.5.8	The <i>with</i> Keyword Revisited	266
9.6	What You Should Know	267
10	It's the BOM! Browser Objects	271
10.1	JavaScript and the Browser Object Model	271
10.1.1	Working with the <i>navigator</i> Object	273
10.1.2	Working with the <i>window</i> Object	285
10.1.3	Creating Timed Events	292
10.1.4	Working with Frames	303
10.1.5	The <i>location</i> Object	315
10.1.6	The <i>history</i> Object	319
10.1.7	The <i>screen</i> Object	322
10.2	What You Should Know	325
11	Working with Forms and Input Devices	327
11.1	The Document Object Model and the Legacy DOM 0	327
11.2	The JavaScript Hierarchy	328
11.2.1	The Document Itself	329
11.3	About HTML Forms	334
11.3.1	Attributes of the <i><form></i> Tag	334
11.4	JavaScript and the <i>form</i> Object	341
11.4.1	Naming Forms and Input Types (Controls) for Forms	342
11.4.2	The Legacy DOM with Forms	345
11.4.3	Naming Forms and Buttons	350
11.4.4	Submitting Fillout Forms	356
11.4.5	The <i>this</i> Keyword	365
11.4.6	The <i>submit()</i> and <i>reset()</i> Methods	368
11.5	Programming Input Devices (Controls)	372
11.5.1	Simple Form Validation	401
11.6	What You Should Know	409
12	Working with Images (and Links)	413
12.1	Introduction to Images	413
12.1.1	HTML Review of Images	414
12.1.2	The JavaScript <i>image</i> Object	416
12.2	Reviewing Links	417
12.2.1	The JavaScript <i>links</i> Object	418

- 12.3 Working with Imagemaps 422
 - 12.3.1 Replacing Images Dynamically with the `src` Property 428
 - 12.3.2 Preloading Images and the `Image()` Constructor 432
 - 12.3.3 Randomly Displaying Images and the `onClick` Event 434
 - 12.3.4 Links with an Image Map and JavaScript 436
- 12.4 Resizing an Image to Fit the Window 438
- 12.5 Introduction to Slideshows 441
 - 12.5.1 A Simple Slideshow with Controls 442
 - 12.5.2 A Clickable Image Slideshow 445
- 12.6 Animation and Timers 449
 - 12.6.1 Changing Image Position 450
 - 12.6.2 Changing Image Height and Width Properties 451
- 12.7 What You Should Know 452

13 Handling Events 455

- 13.1 Introduction to Event Handlers 455
- 13.2 The Inline Model for Handling Events 455
 - 13.2.1 HTML and the Event Handler 456
 - 13.2.2 Setting Up an Event Handler 459
 - 13.2.3 Return Values 461
 - 13.2.4 JavaScript Object Methods and Events 462
- 13.3 Handling a Window or Frame Event 465
 - 13.3.1 The `onLoad` and `onUnload` Events 465
 - 13.3.2 The `onFocus` and `onBlur` Event Handlers 468
 - 13.3.3 The `onResize` Event Handler 472
- 13.4 Handling Mouse Events 474
 - 13.4.1 How to Use Mouse Events 475
 - 13.4.2 Mouse Events and Images—Rollovers 477
 - 13.4.3 Creating a Slideshow with Mouse Events 478
- 13.5 Handling Link Events 481
 - 13.5.1 JavaScript URLs 481
- 13.6 Handling a Form Event 482
 - 13.6.1 Buttons 483
 - 13.6.2 `this` for Forms and `this` for Buttons 484
 - 13.6.3 Forms and the `onClick` Event Handler 486
 - 13.6.4 Forms and the `onFocus` and `onBlur` Event Handlers 487
 - 13.6.5 Forms and the `onChange` Event Handler 489
 - 13.6.6 Forms and the `onSubmit` Event Handler 491
 - 13.6.7 HTML Event Handlers and JavaScript Event Methods 496
 - 13.6.8 The `onError` Event 498
- 13.7 The `event` Object 499
 - 13.7.1 Capturing and Bubbling (Trickle Down and Bubble Up) 500
 - 13.7.2 Event Object Properties 501

- 13.7.3 Using Event Object Properties 503
- 13.7.4 Passing Events to a JavaScript Function 505
- 13.7.5 Mouse Positions 508
- 13.7.6 Key Events 513
- 13.8 The Scripting Model for Handling Events 517
 - 13.8.1 Getting a Reference to the Object 517
- 13.9 What You Should Know 523

14 Introduction to CSS (Cascading Style Sheets) with JavaScript 527

- 14.1 What Is CSS? 527
- 14.2 What Is a Style Sheet? 527
 - 14.2.1 What Is a CSS-Enhanced Browser? 528
 - 14.2.2 How Does a Style Sheet Work? 529
- 14.3 CSS Program Structure 530
 - 14.3.1 Comments 530
 - 14.3.2 Grouping 531
- 14.4 Common Style Sheet Properties 532
 - 14.4.1 Units of Measurement 535
 - 14.4.2 Working with Colors 536
 - 14.4.3 Working with Fonts 539
 - 14.4.4 Working with Text 542
 - 14.4.5 Working with Backgrounds and Images 544
 - 14.4.6 Working with Margins and Borders 547
- 14.5 Types of Style Sheets 550
 - 14.5.1 The Embedded Style Sheet and the `<style>` Tag 550
 - 14.5.2 The Inline Style and the `<style>` Attribute 553
- 14.6 The External Type with a Link 555
 - 14.6.1 The `<link>` Tag 555
 - 14.6.2 Importing with `@import` 557
- 14.7 Creating a Style Class 558
 - 14.7.1 Styling a Simple Table with Class 560
 - 14.7.2 Using a Specific Class Selector 562
- 14.8 The ID Selector and the ID Attribute 564
- 14.9 Overriding or Adding a Style with the `` Tag 566
 - 14.9.1 The `` Tag and the `style` Attribute 567
 - 14.9.2 The `` Tag and the `class` Attribute 568
 - 14.9.3 Inheritance and Contextual Selectors 569
- 14.10 Positioning Elements and Layers 572
 - 14.10.1 Absolute Positioning 573
 - 14.10.2 The `<div>` Container 579
 - 14.10.3 Absolute Positioning 580

- 14.10.4 Relative Positioning 581
- 14.10.5 The *z-index* and Three Dimensions 583
- 14.11 Where Does JavaScript Fit In? 585
 - 14.11.1 What Is DHTML? 585
 - 14.11.2 How JavaScript Views Style Sheets 585
 - 14.11.3 The *style* Object 589
 - 14.11.4 The *className* Property 598
 - 14.11.5 Drop-Down Menus and Tooltips 601
- 14.12 What You Should Know 609

15 The W3C DOM and JavaScript 611

- 15.1 The W3C DOM 611
- 15.2 How the DOM Works with Nodes 612
- 15.3 Nodes 613
 - 15.3.1 Parents and Children 615
 - 15.3.2 Siblings 616
 - 15.3.3 The *nodeName* and *nodeType* Properties 616
 - 15.3.4 The Whitespace Bug 617
- 15.4 Walking with the DOM 618
- 15.5 DOM Inspectors 621
- 15.6 Methods to Shorten the DOM Walk 622
 - 15.6.1 The *document.getElementById()* Method 622
 - 15.6.2 The *document.getElementsByTagName()* Method 625
 - 15.6.3 JavaScript Properties to Represent HTML Attributes 627
- 15.7 Modifying the DOM (Appending, Copying, and Removing Nodes) 629
 - 15.7.1 The *innerHTML* Property and the Element's Content 630
 - 15.7.2 Modifying the Content of an Element 632
 - 15.7.3 Creating New Elements with the DOM 634
 - 15.7.4 Inserting Before a Node 636
 - 15.7.5 Creating Attributes for Nodes 637
 - 15.7.6 DOM Review: Creating a Blog 639
 - 15.7.7 Creating a Table with the DOM 644
 - 15.7.8 Cloning Nodes 648
 - 15.7.9 Removing a Node 653
 - 15.7.10 Scrolling with the Nodes 658
- 15.8 Event Handling and the DOM 661
 - 15.8.1 The HTML Inline Way 661
 - 15.8.2 The Scripting Way 661
 - 15.8.3 The DOM Way 662
 - 15.8.4 Bubbling and Capturing 662
- 15.9 Event Listeners with the W3C Model 668
 - 15.9.1 Adding an Event 668
 - 15.9.2 Registering More Than One Event 670

- 15.9.3 Removing an *EventListener* 673
- 15.9.4 Event Listeners with Microsoft Internet Explorer 676
- 15.9.5 Event Properties Revisited 678
- 15.10 Unobtrusive JavaScript 682
 - 15.10.1 JavaScript Libraries 689
- 15.11 What You Should Know 690
- 16 Cookies 695**
 - 16.1 What Are Cookies? 695
 - 16.1.1 Cookie Ingredients 698
 - 16.1.2 The Attributes of a Cookie 699
 - 16.2 Creating a Cookie with JavaScript 701
 - 16.2.1 The Cookie Object 701
 - 16.2.2 Assigning Cookie Attributes 702
 - 16.2.3 Let's Make Cookies! 704
 - 16.2.4 Retrieving Cookies from a Server 708
 - 16.2.5 Deleting a Cookie 710
 - 16.2.6 Using the Browser to Remove Cookies 713
 - 16.3 What You Should Know 714
- 17 Regular Expressions and Pattern Matching 717**
 - 17.1 What Is a Regular Expression? 717
 - 17.2 Creating a Regular Expression 719
 - 17.2.1 The Literal Way 719
 - 17.2.2 The Constructor Method 720
 - 17.2.3 Testing the Expression 721
 - 17.2.4 Properties of the *RegExp* Object 724
 - 17.3 String Methods Using Regular Expressions 727
 - 17.3.1 The *match()* Method 727
 - 17.3.2 The *search()* Method 729
 - 17.3.3 The *replace()* Method 730
 - 17.3.4 The *split()* Method 731
 - 17.4 Getting Control—The Metacharacters 733
 - 17.4.1 The Dot Metacharacter 736
 - 17.4.2 The Character Class 738
 - 17.4.3 Metasymbols 741
 - 17.4.4 Metacharacters to Repeat Pattern Matches 745
 - 17.4.5 Anchoring Metacharacters 754
 - 17.4.6 Alternation 759
 - 17.5 Form Validation with Regular Expressions 765
 - 17.5.1 Checking for Empty Fields 765
 - 17.5.2 Checking for Numeric Zip Codes 767
 - 17.5.3 Checking for Alphabetic Data 769

- 17.5.4 Removing Extraneous Characters 771
- 17.5.5 Checking for Valid Social Security Numbers 775
- 17.5.6 Checking for Valid Phone Numbers 777
- 17.5.7 Checking for Valid E-Mail Addresses 781
- 17.5.8 Credit Card Validation 783
- 17.5.9 Putting It All Together 791

17.6 What You Should Know 795

18 An Introduction to Ajax (with JSON) 797

18.1 Why Ajax? 797

18.2 Why Is Ajax Covered Last? 798

18.3 The Steps for Creating Ajax Communication 799

18.3.1 Step 1: Create the *XMLHttpRequest* Object 800

18.3.2 Step 2: Initializing the Object 803

18.3.3 Sending the Request to the Server 805

18.3.4 Step 3: Monitoring the State of the Server Response 806

18.3.5 Handling the Response with a Callback Function 808

18.3.6 The Browser Cache Issue 810

18.4 Putting It All Together 812

18.4.1 Using Ajax to Retrieve Text from a File 819

18.4.2 Using Ajax to Retrieve XML from a File 822

18.4.3 Ajax and Forms 826

18.5 Ajax and JSON 834

18.5.1 JSON Data Structures 835

18.5.2 Steps to Use JSON 836

18.5.3 Putting It All Together with JSON 839

18.5.4 Solving the *eval()* Security Problem 843

18.6 Debugging Ajax with Firebug 848

18.6.1 Basic Instructions for Using Firefox 851

18.6.2 What You Should Know 852

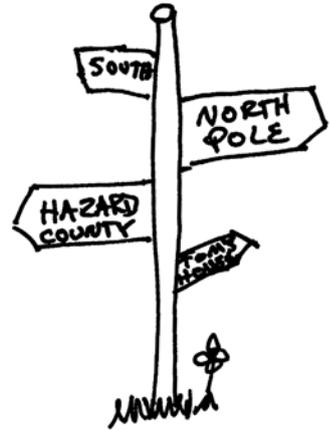
Index 855

Preface

This second edition of *JavaScript by Example* is really more than a new edition; it is a new book! So much has changed since the first edition in 2002, and now with the newfound popularity of Ajax, JavaScript is on a roll! Almost every personal computer has JavaScript installed and running and it is the most popular Web scripting language around, although it comes under different aliases, including Mocha, LiveScript, JScript, and ECMAScript. There are a lot of books out there dedicated to some aspect of the JavaScript language and if you are new to JavaScript, it would be difficult to know where to start. This book is a “one size fits all” edition, dedicated to those of you who need a balance between the technical side of the language and the fun elements, a book that addresses cross-platform issues, and a book that doesn’t expect that you are already a guru before you start. This edition explains how the language works from the most basic examples to the more complex, in a progression that seamlessly leads you from example to example until you have mastered the basics all the way to the more advanced topics such as CSS, the DOM, and Ajax.

Because I am a teacher first, I found that using my first edition worked well in the classroom, but I needed more and better examples to get the results I was looking for. Many of my students have been designers but not programmers, or programmers who don’t understand design. I needed a text that would accommodate both without leaving either group bored or overwhelmed. This huge effort to modernize the first edition went way beyond where I had expected or imagined. I have learned much and hope that you will enjoy sharing my efforts to make this a fun and thorough coverage of a universally popular and important Web programming language.

Under Certain Conditions



6.1 Control Structures, Blocks, and Compound Statements

If you were confronted with the above signpost, you'd have to decide which direction to take. People control their lives by making decisions, and so do programs. In fact, according to computer science books, a good language allows you to control the flow of your program in three ways. It lets you

- Execute a sequence of statements.
- Branch to an alternative sequence of statements, based on a test.
- Repeat a sequence of statements until some condition is met.

Well, then JavaScript must be a good language. We've already used programs that execute a sequence of statements, one after another.

Now we will examine the branching and looping control structures that allow the flow of the program's control to change depending on some conditional expression.

The decision-making constructs (*if*, *if/else*, *if/else if*, *switch*) contain a control expression that determines whether a block of statements will be executed. The looping constructs (*while*, *for*) allow the program to execute a statement block repetitively until some condition is satisfied.

A compound statement or block consists of a group of statements surrounded by curly braces. The block is syntactically equivalent to a single statement and usually follows an *if*, *else*, *while*, or *for* construct.

6.2 Conditionals

Conditional constructs control the flow of a program. If a condition is true, the program will execute a block of statements and if the condition is false, flow will go to an alternate block of statements. Decision-making constructs (*if*, *else*, *switch*) contain a control

expression that determines whether a block of expressions will be executed. If the condition after the *if* is met, the result is true, and the following block of statements is executed; otherwise the result is false and the block is not executed.

FORMAT

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

EXAMPLE

```
if ( age > 21 ){
    alert("Let's Party!");
}
```

The block of statements (or single statement) is enclosed in curly braces. Normally, statements are executed sequentially. If there is only one statement after the conditional expression, the curly braces are optional.

6.2.1 *if/else*

“You better pay attention now, or else . . .” Ever heard that kind of statement before? JavaScript statements can be handled the same way with the *if/else* branching construct. This construct allows for a two-way decision. The *if* evaluates the expression in parentheses, and if the expression evaluates to true, the block after the opening curly braces is executed; otherwise the block after the *else* is executed.

FORMAT

```
if (condition){
    statements1;
}
else{
    statements2;
}
```

EXAMPLE

```
if ( x > y ){
    alert( "x is larger");
}
else{
    alert( "y is larger");
}
```

EXAMPLE 6.1

```
<html>
  <head>
    <title>Conditional Flow Control</title>
  </head>
  <body>
    <h3>
1      <script type="text/javascript">
2        <!-- Hiding JavaScript from old browsers
3          var age=prompt("How old are you? ", "");
4          if( age >= 55 ){
5            document.write("You pay the senior fare! ");
6          }
7          else{
8            document.write("You pay the regular adult fare. ");
9          }
10         <!-->
11       </script>
12     </h3>
13   </body>
14 </html>
```

EXPLANATION

- 1 JavaScript program starts here.
- 2 The prompt dialog box will display the message “*How old are you?*”. Whatever the user types into the box will be stored in the variable *age* (see Figure 6.1).
- 3, 4 If the value of the variable *age* is greater than or equal to 55, line 4 is executed (see Figure 6.2).
- 5 This closing curly brace closes the block of statements following the *if* expression. When there is only one statement in the block, the curly braces are not required.
- 6, 7 The *else* statement, line number 7, is executed if the expression in line 3 is false.
- 8 This tag marks the end of the JavaScript program.



Figure 6.1 The user is prompted for input.



Figure 6.2 If the age entered was greater than 55, this message is displayed.

The Conditional Operator. The conditional operator, called a ternary operator, was discussed in Chapter 5, “Operators.” Because it is often used as a shortcut for the *if/else* conditional statement, it is reviewed again here.

FORMAT

```
conditional expression ? expression : expression
```

EXAMPLE

x ? y : z If *x* evaluates to true, the value of the expression becomes *y*, else the value of the expression becomes *z*

big = (x > y) ? x : y If *x* is greater than *y*, *x* is assigned to variable *big*, else *y* is assigned to variable *big*

An *if/else* statement instead of the conditional statement:

```
if (x > y) {
    big = x;
}
else{
    big = y;
}
```

EXAMPLE 6.2

```
<html>
  <head>
    <title>Conditional Operator</title>
  </head>
  <body bgcolor="lightblue">
    <big>
      <script type="text/javascript">
1         var age = prompt("How old are you? ", "");
2         var price = (age > 55) ? 0 : 7.50;
```

EXAMPLE 6.2 (CONTINUED)

```

3       alert("You pay $" + price + 0);
        </script>
        </big>
    </body>
</html>

```

EXPLANATION

- 1 The user is prompted for input. The value he or she enters in the prompt box is assigned to the variable *age*.
- 2 If the value of *age* is greater than 55, the value to the right of the ? is assigned to the variable *price*; if not, the value after the : is assigned to the variable *price*.
- 3 The alert dialog box displays the value of the variable *price*.

6.2.2 if/else if

“If you’ve got \$1, we can go to the Dollar Store; else if you’ve got \$10, we could get a couple of movies; else if you’ve got \$20 we could buy a CD . . . else forget it!” JavaScript provides yet another form of branching, the *if/else if* construct. This construct provides a multiway decision structure.

FORMAT

```

if (condition) {
    statement(s);
}
else if (condition) {
    statement(s);
}
else if (condition) {
    statement(s);
}
else{
    statement(s);
}

```

If the first conditional expression following the *if* keyword is true, the statement or block of statements following the expression is executed and control starts after the final *else* block. Otherwise, if the conditional expression following the *if* keyword is false, control branches to the first *else if* and the expression following it is evaluated. If that expression is true, the statement or block of statements following it are executed, and if false, the next *else if* is tested. All *else ifs* are tested and if none of their expressions are true, control goes to the *else* statement. Although the *else* is not required, it normally serves as a default action if all previous conditions were false.

EXAMPLE 6.3

```

<html>
  <head>
    <title>Conditional Flow Control</title>
  </head>
  <body>
    <h2>
1     <script type="text/javascript">
      <!--
2       var age=eval( prompt("How old are you? ", ""));
3       if( age > 0 && age <= 12 ){
4         alert("You pay the child's fare. ");
5         }
6         else if( age > 12 && age < 60 ){
7           alert("You pay the regular adult fare. ");
8         }
9         else {
10          alert("You pay the senior fare! ");
11        }
12      </script>
    </h2>
  </body>
</html>

```

EXPLANATION

- 1 JavaScript program starts here.
- 2 The prompt dialog box will display the message “*How old are you?*”. Whatever the user types into the box will be converted to a number by the `eval()` method and then stored in the variable `age`.
- 3, 4 If the value of the variable `age` is greater than 0 and `age` is also less than or equal to 12, then line 4 is executed and the program continues at line 8.
- 5, 6 If the expression on line 3 is false, the JavaScript interpreter will test this line, and if the `age` is greater than 12 and also less than 60, the block of statements that follow will be executed and control goes to line 8. You can have as many *else ifs* as you like.
- 7 The *else* statement, line number 7, is executed if all of the previous expressions test false. This statement is called the default and is not required.
- 8 This tag marks the end of the JavaScript program.

6.2.3 switch

The *switch* statement is an alternative to *if/else if* conditional construct (commonly called a “case statement”) and may make the program more readable when handling multiple options.

FORMAT

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

EXAMPLE

```
switch (color){
  case "red":
    alert("Hot!");
    break;
  case "blue":
    alert("Cold.");
    break;
  default:
    alert("Not a good choice.");
    break;
}
```

The value of the *switch* expression is matched against the expressions, called labels, following the *case* keyword. The *case* labels are constants, either string or numeric. Each label is terminated with a colon. The default label is optional, but its action is taken if none of the other cases match the *switch* expression. After a match is found, the statements after the matched label are executed for that case. If none of the cases are matched, the control drops to the *default* case. The default is optional. If a *break* statement is omitted, all statements below the matched label are executed until either a *break* is reached or the entire *switch* block exits.

EXAMPLE 6.4

```
<html>
  <head>
    <title>The Switch Statement</title>
  </head>
  <body>
```

Continues

EXAMPLE 6.4 (CONTINUED)

```

<script type="text/javascript">
  <!--
1    var day_of_week=Math.floor((Math.random()* 7)+1);
      // Get a random number between 1 and 7
      // Monday is 1, Tuesday is 2, etc.
2    switch(day_of_week){
3      case 1:
4      case 2:
5      case 3:
6      case 4:
7        alert("Business hours Monday through Thursday are from
8          9am to 10pm");
9        break;
10     case 5:
11       alert("Business hours on Friday are from 9am to 6pm");
12       break;
13     case 6:
14       alert("Business hours on Saturday are from
15         11am to 3pm");
16       break;
17     default:
18       alert("We are closed on Sundays and holidays");
19       break;
20   }
  //-->
</script>
</body>
</html>

```

EXPLANATION

- 1 The random number function generates a random number between 1 and 7 inclusive when the script is executed. The random number is stored in a variable called *day_of_week*.
- 2 The *day_of_week* value of the *switch* expression is matched against the values of each of the *case* labels below.
- 3 The first *case* that is tested is 1. If the random number is 1, the message “*Business hours Monday through Thursday are from 9am to 10pm*” will be displayed in the alert dialog box. The same is true for case 2, 3, and 4.
- 4 This statement is executed if case 1, 2, 3, or 4 are matched. Note there are no *break* statements associated with any of these 4 case statements. Program control just drops from one case to the next, and if cases 1, 2, 3, or 4 are not matched, execution control goes to the next case (case 5) for testing.
- 5 The *break* statement causes program control to continue after line 8. Without it, the program would continue executing statements into the next *case*, “*yellow*”, and continue doing so until a *break* is reached or the *switch* ends—and we don’t want that. The *break* statement sends control of the program to line 8.

EXPLANATION (CONTINUED)

- 6 The default statements are executed if none of the cases are matched.
- 7 This final *break* statement is not necessary, but is good practice in case you should decide to replace the *default* with an additional *case* label.
- 8 The final curly brace ends the *switch* statement. Figure 6.3 displays examples of the output.

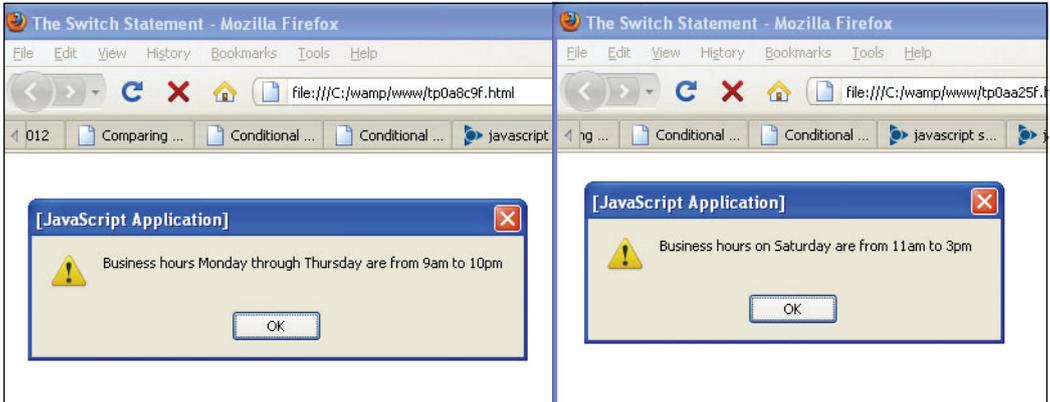


Figure 6.3 A random number between 1 and 7 determines which case is matched and executed.

6.3 Loops

Loops are used to execute a segment of code repeatedly until some condition is met. JavaScript's basic looping constructs are

- *while*
- *for*
- *do/while*

6.3.1 The *while* Loop

The *while* statement executes its statement block as long as the expression after the while evaluates to true; that is, nonnull, nonzero, nonfalse. If the condition never changes and is true, the loop will iterate forever (infinite loop). If the condition is false, control goes to the statement right after the closing curly brace of the loop's statement block.

The *break* and *continue* functions are used for loop control.

FORMAT

```
while (condition) {
    statements;
    increment/decrement counter;
}
```

EXAMPLE 6.5

```
<html>
  <head>
    <title>Looping Constructs</title>
  </head>
  <body>
    <h2>While Loop</h2>
    <font size="+2">
1     <script type="text/javascript">
2       var i=0;    // Initialize loop counter
3       while ( i < 10 ){      // Test
4         document.writeln(i);
5         i++;    // Increment the counter
6       }    // End of loop block
7     </script>
    </font>
  </body>
</html>
```

EXPLANATION

- 1 The JavaScript program starts here.
- 2 The variable *i* is initialized to 0.
- 3 The expression after the *while* is tested. If *i* is less than 10, the block in curly braces is entered and its statements are executed. If the expression evaluates to false, (i.e., *i* is not less than 10), the loop block exits and control goes to line 6.
- 4 The value of *i* is displayed in the browser window (see Figure 6.4).
- 5 The value of *i* is incremented by 1. If this value never changes, the loop will never end.
- 6 This curly brace marks the end of the *while* loop's block of statements.
- 7 The JavaScript program ends here.



Figure 6.4 Output from Example 6.5.

6.3.2 The *do/while* Loop

The *do/while* statement executes a block of statements repeatedly until a condition becomes false. Owing to its structure, this loop necessarily executes the statements in the body of the loop at least once before testing its expression, which is found at the bottom of the block. The *do/while* loop is supported in Mozilla/Firefox and Internet Explorer 4.0, JavaScript 1.2, and ECMAScript v3.

FORMAT

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

EXAMPLE 6.6

```
<html>
  <head>
    <title>Looping Constructs</title>
  </head>
  <body>
    <h2>Do While Loop</h2>
    <font size="+2">
      <script type="text/javascript">
1         var i=0;
2         do{
3           document.writeln(i);
4           i++;
5         } while ( i < 10 )
      </script>
    </font>
  </body>
</html>
```

EXPLANATION

- 1 The variable *i* is initialized to 0.
- 2 The *do* block is entered. This block of statements will be executed before the *while* expression is tested. Even if the *while* expression proves to be false, this block will be executed the first time around.
- 3 The value of *i* is displayed in the browser window (see Figure 6.5).
- 4 The value of *i* is incremented by 1.
- 5 Now, the *while* expression is tested to see if it evaluates to true (i.e., is *i* less than 10?). If so, control goes back to line 2 and the block is re-entered.

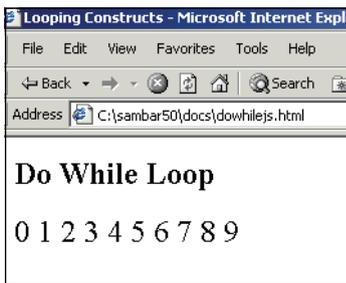


Figure 6.5 Output from Example 6.6, the *do/while* loop.

6.3.3 The *for* Loop

The *for* loop consists of the *for* keyword followed by three expressions separated by semicolons and enclosed within parentheses. Any or all of the expressions can be omitted, but the two semicolons cannot. The first expression is used to set the initial value of variables and is executed just once, the second expression is used to test whether the loop should continue or stop, and the third expression updates the loop variables; that is, it increments or decrements a counter, which will usually determine how many times the loop is repeated.

FORMAT

```
for(Expression1;Expression2;Expression3)
{statement(s);}
for(initialize; test; increment/decrement)
{statement(s);}
```

The preceding format is equivalent to the following *while* statement:

```
Expression1;
while( Expression2 )
{ Block; Expression3};
```

EXAMPLE 6.7

```

<html>
  <head>
    <title>Looping Constructs</title>
  </head>
  <body>
    <h2>For Loop</h2>
    <font size="+2">
      <script type="text/javascript">
1         for( var i = 0; i < 10; i++ ){
2           document.write(i);
3         }
      </script>
    </font>
  </body>
</html>

```

EXPLANATION

- 1 The *for* loop is entered. The expression starts with step 1, the initialization of the variable *i* to 0. This is the only time this step is executed. The second expression, step 2, tests to see if *i* is less than 10, and if it is, the statements after the opening curly brace are executed. When all statements in the block have been executed and the closing curly brace is reached, control goes back into the *for* expression to the last expression of the three. *i* is now incremented by one and the expression in step 2 is retested. If true, the block of statements is entered and executed.
- 2 The value of *i* is displayed in the browser window (see Figure 6.6).
- 3 The closing curly brace marks the end of the *for* loop.



Figure 6.6 Output from Example 6.7.

6.3.4 The *for/in* Loop

The *for/in* loop is like the *for* loop, except it is used with JavaScript objects. Instead of iterating the statements based on a looping condition, it operates on the properties of an object. This loop is discussed in Chapter 9, “JavaScript Core Objects,” and is only mentioned here in passing, because it falls into the category of looping constructs.

6.3.5 Loop Control with *break* and *continue*

The control statements, *break* and *continue*, are used to either break out of a loop early or return to the testing condition early; that is, before reaching the closing curly brace of the block following the looping construct.

Table 6.1 Control Statements

Statement	What It Does
<i>break</i>	Exits the loop to the next statement after the closing curly brace of the loop's statement block.
<i>continue</i>	Sends loop control directly to the top of the loop and re-evaluates the loop condition. If the condition is true, enters the loop block.

EXAMPLE 6.8

```

<html>
  <head>
    <title>Looping Constructs</title>
  </head>
  <body>
1    <script type="text/javascript">
2      while(true) {
3        var grade=eval(prompt("What was your grade? ", ""));
4        if (grade < 0 || grade > 100) {
5          alert("Illegal choice!");
6          continue; // Go back to the top of the loop
7        }
8        if(grade > 89 && grade < 101)
9          {alert("Wow! You got an A!");}
10       else if (grade > 79 && grade < 90)
11         {alert("You got a B");}
12       else if (grade > 69 && grade < 80)
13         {alert("You got a C");}
14       else if (grade > 59 && grade < 70)
15         {alert("You got a D");}
16       else {alert("Study harder. You Failed.");}
17       answer=prompt("Do you want to enter another grade? ", "");
18       if(answer != "yes"){
19         break; // Break out of the loop to line 12
20       }
21     }
22   </script>
23 </body>
24 </html>

```

EXPLANATION

- 1 The JavaScript program starts here.
- 2 The *while* loop is entered. The loop expression will always evaluate to true, causing the body of the loop to be entered.
- 3 The user is prompted for a grade, which is assigned to the variable *grade*.
- 4 If the variable *grade* is less than 0 or more than 100, “*Illegal choice*” is printed.
- 5 The *continue* statement sends control back to line 2 and the loop is re-entered, prompting the user again for a grade.
- 6 If a valid grade was entered, and it is greater than 89 and less than 101, the grade “A” is displayed (see Figure 6.7).
- 7 Each *else/if* branch will be evaluated until one of them is true.
- 8 If none of the expressions are true, the *else* condition is reached and “*You Failed*” is displayed.
- 9 The user is prompted to see if he or she wants to enter another grade.
- 10, 11 If the answer is not *yes*, the *break* statement takes the user out of the loop, to line 12.

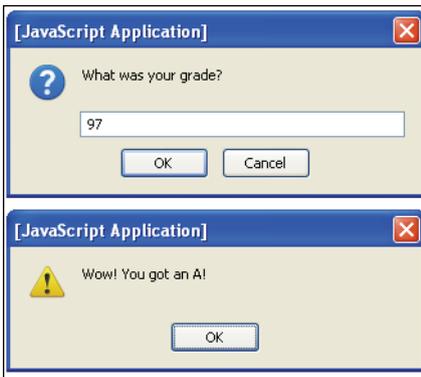


Figure 6.7 The user enters a grade, clicks OK, and gets another alert box.

6.3.6 Nested Loops and Labels

Nested Loops. A loop within a loop is a nested loop. A common use for nested loops is to display data in rows and columns. One loop handles the rows and the other handles the columns. The outside loop is initialized and tested, the inside loop then iterates completely through all of its cycles, and the outside loop starts again where it left off. The inside loop moves faster than the outside loop. Loops can be nested as deeply as you wish, but there are times when it is necessary to terminate the loop owing to some condition.

EXAMPLE 6.9

```

<html>
  <head>
    <title>Nested loops</title>
  </head>
  <body>
    <script type="text/javascript">
      <!-- Hiding JavaScript from old browsers
1      var str = "@";
2      for ( var row = 0; row < 6; row++){
3          for ( var col=0; col < row; col++){
              document.write(str);
          }
4          document.write("<br />");
          }
          //-->
    </script>
  </body>
</html>

```

EXPLANATION

- 1 The variable *str* is assigned a string “@”.
- 2 The outer *for* loop is entered. The variable *row* is initialized to 0. If the value of *row* is less than 6, the loop block (in curly braces) is entered (i.e., go to line 3).
- 3 The inner *for* loop is entered. The variable *col* is initialized to 0. If the value of *col* is less than the value of *row*, the loop block is entered and an @ is displayed in the browser. Next, the value of *col* will be incremented by 1, tested, and if still less than the value of *row*, the loop block is entered, and another @ displayed. When this loop has completed, a row of @ symbols will be displayed, and the statements in the outer loop will start up again.
- 4 When the inner loop has completed looping, this line is executed, producing a break in the rows (see Figure 6.8).



```

@
@@
@@@
@@@@
@@@@@

```

Figure 6.8 Nested loops: rows and columns. Output from Example 6.9.

Labels. Labels allow you to name control statements (*while*, *do/while*, *for*, *for/in*, and *switch*) so that you can refer to them by that name elsewhere in your program. They can be named the same as any other legal identifier that is not a reserved word. By themselves, labels do nothing. Labels are optional, but are often used to control the flow of a loop. A label looks like this, for example:

```
topOfLoop:
```

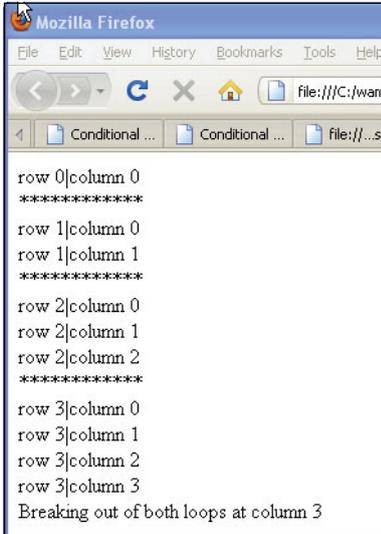
Normally, if you use loop-control statements such as *break* and *continue*, the control is directed to the innermost loop. There are times when it might be necessary to switch control to some outer loop. This is where labels most often come into play. By prefixing a loop with a label, you can control the flow of the program with *break* and *continue* statements as shown in Example 6.10. Labeling a loop is like giving the loop its own name.

EXAMPLE 6.10

```
<script type="text/javascript">
1  outerLoop: for ( var row = 0; row < 10; row++){
2      for ( var col=0; col <= row; col++){
3          document.write("row "+ row +"|column " + col, "<br />");
4          if(col==3){
5              document.write("Breaking out of outer loop at column
6                  " + col +"<br />");
7              break outerLoop;
8          }
9      }
10     document.write("*****<br />");
11 } // end outer loop block
</script>
```

EXPLANATION

- 1 The label *outerLoop* labels the *for* loop that follows it. It's like giving the *for* loop its own name so that it can be referenced by that name later.
- 2 This is a nested *for* loop. As the program executes the row and column numbers are displayed.
- 3 If the expression is true, the *break* statement, with the label, causes control to go to line 8; it breaks out of the *outer*: loop. A *break* statement without a label would cause the program to exit just the loop to which it belongs.
- 4 The value of *row* and *col* are displayed as the inner loop iterates.
- 5 The *break* statement with the label causes control to go to line 8.
- 6 Each time the inner loop exits, this row of stars will be printed (see Figure 6.9). Notice that the row of stars is not printed when the loop is exited on line 5.
- 7 The closing curly brace closes the outer *for* loop block on line 1.



```

row 0|column 0
*****
row 1|column 0
row 1|column 1
*****
row 2|column 0
row 2|column 1
row 2|column 2
*****
row 3|column 0
row 3|column 1
row 3|column 2
row 3|column 3
Breaking out of both loops at column 3

```

Figure 6.9 Using a label with a loop.

6.4 What You Should Know

“Two roads diverged in a wood, and I—” wrote Robert Frost. This chapter was about making decisions about the flow of your program, what road to take, how to repeat a sequence of statements, and how to stop the repetition. At this point, you should understand:

1. How to use conditional constructs to control the flow of your program; *if/else*, *switch*, and so on.
2. What a block is and when to use curly braces.
3. How and why you would use a *switch* statement.
4. How the *while* and the *do/while* loops differ.
5. How to use a *for* loop.
6. How to use *break* and *continue* with loops.
7. The purpose of nested loops.
8. How to make an infinite loop and how to get out of it.
9. The purpose of labels in loops.
10. How *else/ifs* work.

Exercises

1. Create a *while* loop that displays numbers as: 10 9 8 7 6 5 4 3 2 1. Put the numbers in HTML table cells.
2. Ask the user what the current hour is. If the hour is between 6 and 9 a.m., tell the user, "Breakfast is served." If the hour is between 11 a.m. and 1 p.m., tell the user, "Time for lunch." If the hour is between 5 and 8 p.m., tell the user, "It's dinner time." For any other hours, tell the user, "Sorry, you'll have to wait, or go get a snack."
3. Create a conversion table using the following formula:

$$C = (F - 32) / 1.8;$$

Start with a Fahrenheit temperature of 20 degrees and end with a temperature of 120 degrees; use an increment value of 5. The table will have two columns, one for Fahrenheit temperature values and one for those same temperatures converted to Celsius.

4. Ask the user for the name of the company that developed the JavaScript language. Alert the user when he or she is wrong, and then keep asking the user until he or she gets the correct answer. When the user gets it right, confirm it.
5. Use a *switch* statement to rewrite the following JavaScript code. Prompt the user for the number of a month rather than setting it to 8.

```
<script type=text/javascript>
  month = 8;

  if (month == 1) {
    alert("January");
  }
  else if (month == 2) {
    alert("February");
  }
  else if (month == 3) {
    alert("March");
  }
  else if (month == 4) {
    alert("April");
  }
  else if (month == 5) {
    alert("May");
  }
}
```

```
else if (month == 6) {
    alert("June");
}
else if (month == 7) {
    alert("July");
}
else if (month == 8) {
    alert("August");
}
else if (month == 9) {
    alert("September");
}
else if (month == 10) {
    alert("October");
}
else if (month == 11) {
    alert("November");
}
else if (month == 12) {
    alert("December");
}
else{
    alert("Invalid month");
}
</script>
```

6. Consider the following example:

```
var start_time = (day == weekend) ? 12 : 9;
```

Rewrite the conditional statement using an *if/else* construct.

Index

Symbols

& (ampersand)

- bitwise AND (&) operator, 110, 111–12
- logical AND (&&) operator, 101–3

<> (angle brackets)

- in bitwise shift operators, 110–12
- in comparison operators, 95–96, 98–100
- greater-than (>) operator, 95, 98, 99
- greater-than or equal-to (>=) operator, 96, 98, 99
- left shift (<<) operator, 110–12
- less-than (<) operator, 96, 98, 99
- less-than or equal-to (<=) operator, 96, 98, 99
- right shift (>>) operator, 110–12
- zero-fill right shift (>>>) operator, 110–12

* (asterisk) as multiplication operator, 88

\ (backslash) and regular expressions, 733, 734, 735

^ (bitwise XOR operator), 110, 111–12

{ } (curly braces) in function statements, 144, 145

= (equal sign)

- as assignment operator, 84, 90–91
- in comparison operators, 95–98
- equal-to (==) operator, 41, 58, 96–97
- identical-to (===) operator, 96, 97
- not-equal-to (!=) operator, 95
- not-identical-to (!==) operator, 96, 97
- and operator precedence, 84, 85

/ (forward slash)

- as division operator, 88
- in regular expressions, 717, 720

! (logical NOT operator), 101, 105–6

- (minus sign)

- auto-decrement (-- operator), 91–94
- bitwise NOT (~) operator, 110, 111–12
- as subtraction operator, 88

| (OR operator, bitwise), 110, 111–12

|| (OR operator, logical), 101, 103–5

() (parentheses)

- following method names, 13, 38, 180, 333
- in function statements, 144, 146, 156, 518
- and operator precedence, 84, 85
- and regular expressions, 736, 761, 762, 772–73, 777–80, 781, 790

+ (plus sign)

- as addition operator, 88
- auto-increment (++) operator, 91–94
- as concatenation operator, 66, 94–95

" (quote marks, double), 54

' (quote marks, single), 54

[] (square brackets) for notation, 213, 216, 222–23, 353

A

abort() method, *XMLHttpRequest* object, 802

abs() method, *Math* object, 242

absolute positioning, CSS elements, 573–79

- accessKey property
 - checkbox object, 396
 - password object, 378
 - radio object, 393
 - text object, 373
 - textarea object, 382
- acos() method, *Math* object, 242
- action HTML `<form>` tag attribute, 335, 356, 826, 827
- action property, *forms* object, 349
- ActiveX, 280–82, 800
- addEventListener()* method, in W3C DOM event model, 668–69
- Ajax (Asynchronous JavaScript and XML)
 - defined, 5–6, 797
 - existing application examples, 6–7, 797–98
 - overview, 5–7, 797
 - PHP server-side script example, 813–18
 - reasons for using with forms, 826–34
 - retrieving text from file example, 819–22
 - retrieving XML from file example, 822–26
 - role in Web page cycle, 6
 - role of *XMLHttpRequest* object, 800–810
 - steps in creating communications, 799–811
- alert()* method, *window* object, 73–76, 286
- alink* HTML `<body>` tag attribute, 329
- alinkColor* property, *document* object, 330
- alt* property
 - checkbox object, 396
 - password object, 378
 - radio object, 393
 - text object, 373
- altKey* property, *event* object, 501, 502
- anchor()* method, *String* object, 251
- AND operators
 - bitwise (`&`), 110, 111–12
 - logical (`&&`), 101–3
- angle brackets (`<>`)
 - in comparison operators, 95–96, 98–100
 - greater-than (`>`) operator, 95, 98, 99
 - greater-than or equal-to (`>=`) operator, 96, 98, 99
 - less-than (`<`) operator, 96, 98, 99
 - less-than or equal-to (`<=`) operator, 96, 98, 99
- animation, JavaScript, 449–52
- appName* property, *navigator* object, 273
- appendChild()* DOM method, 615, 630, 644
- apply()* property, *Function* object, 265
- appName* property, *navigator* object, 273
- appVersion* property, *navigator* object, 273
- `<area>` tag, 414, 424
- arguments, passing to functions, 146–48
- arithmetic operators, 88–89
- Array()* constructor, 177, 214–16, 218, 219, 220
- array literals, 216–17
- arrays
 - associative, 213, 214, 221–23
 - creating by using literal notation, 216–17
 - creating by using *new* keyword, 214–16
 - length* property, 219, 220
 - methods, 227–33
 - nested, 223–27
 - numeric compared with associative, 213–14
 - overview, 213–14
 - populating, 217–19
 - properties, 219–21
 - style sheets in, 585–88
 - two-dimensional, 223–25
 - types, 213
- asin()* method, *Math* object, 242
- assignment operator (`=`), 84, 90–91
- assignment statements, 84
- associative arrays, 213, 214, 221–23
- associativity, operator, 84–88
- asterisk (`*`) as multiplication operator, 88
- Asynchronous JavaScript and XML. *See* Ajax (Asynchronous JavaScript and XML)
- atan()* method, *Math* object, 242
- atan2()* method, *Math* object, 242
- Attribute DOM object, defined, 613
- attribute nodes, DOM, defined, 614
- attributes, HTML `<script>` tag, 36–37
- auto-decrement operator, 91–94
- auto-increment operator, 91–94
- availDepth* property, *screen* object, 323
- availHeight* property, *screen* object, 322
- availLeft* property, *screen* object, 322

availTop property, *screen* object, 323
availWidth property, *screen* object, 323

B

back() method, *history* object, 319
background HTML `<body>` tag attribute, 329
background-attachment CSS property, 533, 544
background-color CSS property, 533, 536
background-image CSS property, 533, 544
background-position CSS property, 533, 544
background-repeat CSS property, 533, 544
backgrounds, in CSS, 533, 544–46
backslash (\) and regular expressions, 733, 734, 735
behavior, role in Web page design, 10
bgcolor HTML `<body>` tag attribute, 329
bgColor property, *document* object, 330
big() method, *String* object, 251
binary number system, 109
bit, defined, 109
bitwise AND (&), 110, 111–12
bitwise NOT (-), 110, 111–12
bitwise operators, 109–12
bitwise OR (|), 110, 111–12
bitwise XOR (^), 110, 111–12
blink() method, *String* object, 251
blog entries, adding, 639–45
blur() event method, 463, 470–71
blur() method

- checkbox* object, 397
- frame* object, 308
- password* object, 379
- radio* object, 393
- select* object, 386
- text* object, 374
- textarea* object, 382
- window* object, 286

`<body>` tag, HTML

- attributes defining *document* object, 329
- compared with `<frameset>` tag, 303
- as container, 547
- in DOM tree-structure, 613, 614
- and event handlers, 459
- and JavaScript code, 20

bold() method, *String* object, 252
Boolean data type, 56–57, 63, 112–19. *See also* comparison operators
Boolean() function, 113–14
Boolean object, 246, 263–64
border property, *image* object, 417
border-bottom CSS property, 534, 547
border-bottom-width CSS property, 534, 547
border-color CSS property, 534, 547
border-left CSS property, 534, 547
border-left-width CSS property, 534, 548
border-right CSS property, 534, 548
border-right-width CSS property, 534, 548
border-style CSS property, 534, 548
border-top CSS property, 534, 548
border-top-width CSS property, 534, 548
border-width CSS property, 534, 548
borders, in Cascading Style Sheets, 534–35, 547–49
bottom property, in positioning CSS elements, 573
break statement, 136–37
Browser Object Model (BOM)

- compared with Document Object Model (DOM), 271–72
- defined, 271
- hierarchy, 271
- window* object, 73, 179, 285–87

browser sniffers, 15–16, 276–78
Browser Wars, 12
BrowserDetect object, 277
browsers. *See also* Firefox; Internet Explorer; Netscape; Opera browser; Safari

- cache issue, 810–11
- CSS-enhanced, 528–29
- dividing window into frames, 303–14
- DOM inspectors, 621, 622
- executing JavaScript programs, 30–33
- interpreters, 15
- invoking error consoles, 41
- JavaScript in, 1–2, 15, 18–19
- navigator* object detection properties, 276–78
- older, and JavaScript, 47–50
- overview, 15–19

browsers (*Continued*)

- passing events to JavaScript functions, 505–8
- sniffers for, 15–16, 276–78
- support for DOM specification, 13
- support for JavaScript versions, 16–17
- bubbles* property, *event* object, 502, 510, 511, 679
- bubbling, 500, 662–68
- button*, HTML `<form>` tag element. *See also* *radio*, HTML `<form>` tag element
 - attributes, 337
 - description, 337, 353
 - properties, 353
- button* object. *See also* *radio* object
 - event handlers, 483
 - and *this* keyword, 484–85
 - as triggering device, 367–68
- button* property
 - event* object, 501, 502
 - forms* object, 348, 349
- byte, defined, 109

C

- call()* property, *Function* object, 265
- callback functions, 808–9
- Camino browser, 13, 14
- cancelable* property, *event* object, 502, 679
- cancelBubble* property, *event* object, 501, 665–68
- capturing, 500, 662–68
- Cascading Style Sheets (CSS)
 - absolute positioning of elements, 573–79
 - and backgrounds, 533, 544–46
 - and borders, 534–35, 547–49
 - and colors, 533, 536–38
 - comments in, 530
 - common properties, 532–49
 - defining, 550–54
 - embedded, 550–53
 - example, 529–30
 - external, 550, 555–58
 - and fonts, 533, 539–41
 - how they work, 529–30
 - id* attribute, 565–66
 - ID selectors, 564–65
 - and images, 544–46
 - inline, 550, 553–54
 - and margins, 534–35, 547–49
 - multiple, in arrays, 585–88
 - order of precedence, 558
 - overview, 527–28
 - positioning elements and layers, 572–84
 - program structure, 530–32
 - role of grouping, 531–32
 - role of JavaScript, 585–608
 - role of `` tag, 566–72
 - rules for, 529–30
 - style classes for, 558–62
 - text properties, 533–34, 542–44
 - types, 550–54
 - units of measurement, 535–36
- case sensitivity, in JavaScript, 33, 177, 517
- case statements. *See if/else if* statements
- catch* statement, 800. *See also try/catch* statements
- cssRules* array, W3C, 585
- ceil()* method, *Math* object, 242, 244
- Champeon, Steve, 683
- character class, in regular expressions, 738–41
- characters. *See metacharacters*
- charAt()* method, *String* object, 253, 257, 258
- charCode* property, *event* object, 502, 513
- charset* property, *links* object, 419
- checkbox*, HTML `<form>` tag element
 - attributes, 337
 - description, 337, 353
 - properties, 353
- checkbox* object
 - event handlers, 483
 - JavaScript code example, 397–400
 - in JavaScript hierarchy, 396
 - methods, 397
 - overview, 395
 - properties, 396
- checkbox* property, *forms* object, 349
- checked* property
 - checkbox* object, 396
 - radio* object, 393
- child nodes, DOM, defined, 614

- Chrome. *See* Google Chrome
- circle* shape, image map, 425
- class attributes, 558–59, 562
- class HTML `` tag attribute, 566, 568–69
- class properties, *RegExp* object, 724, 725
- class selectors, 558, 562, 563
- classes
 - defined, 182
 - defining for styles, 558–60
 - functions as, 182
 - in object-oriented languages, 196
 - simulated, extending with prototypes, 196
- className* property, as HTML attribute, 598–601, 627
- clear* CSS property, 534
- clear()* method, *document* object, 333
- clearInterval()* method
 - frame* object, 308
 - window* object, 286
- clearTimeout()* method
 - frame* object, 308
 - window* object, 286
- click* event. *See onClick* event handler
- click()* event method, 463
- click()* method
 - checkbox* object, 397
 - radio* object, 393
- client-side JavaScript, defined, 2
- clientX* property, *event* object, 501, 502, 508
- clientY* property, *event* object, 501, 502, 508
- clip* property, in positioning CSS elements, 573
- cloneNode()* DOM method, 615, 630, 648–53
- cloning nodes, 648–53
- close()* method
 - document* object, 333
 - window* object, 286
- closed* property, *window* object, 285
- closing windows, 287, 290
- closures, 158–61
- color* CSS property, 533, 536
- colorDepth* property, *screen* object, 323, 537
- colors, CSS, 533, 536–38
- cols* property, *textarea* object, 382
- comma operator (`,`), 85, 120
- comments
 - in Cascading Style Sheets, 530
 - in JavaScript, 35, 47–49
- comparison operators
 - equal-to (`==`) operator, 41, 58, 96–97
 - greater-than (`>`) operator, 95, 98, 99
 - greater-than or equal-to (`>=`) operator, 96, 98, 99
 - identical-to (`===`) operator, 96, 97
 - less-than (`<`) operator, 96, 98, 99
 - less-than or equal-to (`<=`) operator, 96, 98, 99
 - not-equal-to (`!=`) operator, 95
 - not-identical-to (`!==`) operator, 96, 97
- compile-time errors, 40
- complete* property, *image* object, 417
- composite data types, defined, 59
- concat()* method
 - arrays, 227, 228
 - String* object, 253
- concatenating
 - plus-sign (`+`) operator, 66, 94–95
 - strings, 37, 56
 - strings and variables together, 66–67
- conditional constructs
 - if/else*, 124–27
 - if/else if*, 127–28
 - overview, 123–24
 - switch*, 128–31
- conditional operator, 108–9, 126–27
- confirm()* method, *window* object, 78–80, 286
- const* keyword, 67–69
- constants, 67–69
- constructor* property
 - Array* object, 219, 220
 - defined, 196, 204
- constructors
 - built-in, 177
 - creating objects with, 177–78
 - defined, 177
 - Function()* constructor, 264
 - Image()* constructor, 432–34
 - role of *new* operator, 177–78, 182
- containers, for CSS elements, 528, 547
- content, role in Web page design, 9
- continue* statement, 136–37

controls, form, programming, 372–401
 converting data types, 62, 63, 64–65, 112–19
cookie property, *document* object, 330, 701–4
 cookies
 assigning attributes, 702–4
 attribute overview, 699–701
 creating with JavaScript, 704–8
 defined, 695
 deleting by setting expiration date, 710–13
 deleting by using browser to remove from
 hard drive, 713
 expiration dates, 700
 on Firefox, 697
 on Internet Explorer, 697, 698
 limiting to local browsers, 698–99
 on Opera, 698
 origin, 695
 overview, 695–98
 retrieving from server, 708–10
 session compared with persistent, 695–96
 core objects
 arrays as, 213–33
 Boolean object, 246, 263–64
 Date object, 234–41
 Function object, 246, 265–66
 Math object, 241–46
 Number object, 246, 247, 259–63
 overview, 213
 String object, 246, 247–59
 wrapper objects, 246–64
cos() method, *Math* object, 242
 Crawford, Douglas, 159
createElement() DOM method, 634–35, 644
 credit card numbers, validating, 783–90
 CSS. *See* Cascading Style Sheets (CSS)
ctrlKey property, *event* object, 501, 502
 curly braces ({} in function statements, 144,
 145
current property, *history* object, 319
currentTarget property, *event* object, 502, 679

D

data property, *event* object, 503
 data types
 boolean, 56–57

 composite, 59
 converting, 62, 63, 64–65, 112–19
 JavaScript compared with Java, 2–3
 numeric, 53–54
 overview, 53
 primitive, 53–59
 string, 54–56
Date() constructor, 177
Date object
 customizing with *prototype* property,
 240–41
 manipulating dates and times, 238–40
 methods, 235–38
 overview, 234–35
 debugging
 function errors, 166–68
 tools for, 41–46
 types of errors, 40–41
 using Firefox tools, 41–44
 using Internet Explorer tools, 44–45
 using *JavaScript*: URL protocol, 46
 using *try/catch* and *throw* exception
 handlers, 168–72
 decimal number system, 109
 declaration blocks, in style sheet rules,
 529–30, 531, 532
 declaring variables, 60–62
default shape, image map, 425
defaultChecked property
 checkbox object, 396
 radio object, 393
defaultStatus property, *window* object,
 285
defaultValue property
 password object, 378
 text object, 373
 textarea object, 382
delete operator, 120
 derived classes. *See* subclasses
description property
 mimeType object, 283
 plugin object, 279
 DHTML (Dynamic HTML), 585
 dialog boxes, creating, 73–80
dir property, as HTML attribute, 627

- disabled* property
 - elements* object, 350
 - links* object, 419
 - select* object, 385
 - textarea* object, 382
 - display* property, in positioning CSS
 - elements, 573
 - `<div>` containers, 547, 579–80, 581, 596–98, 658
 - Document DOM object, defined, 613
 - document* object
 - defined, 328
 - JavaScript hierarchy, 328–32
 - list of HTML `<body>` tag attributes, 329
 - methods, 333–34
 - overview, 329
 - properties, 330
 - properties in JavaScript examples, 331–33
 - as property of *window* object, 329
 - role in HTML documents, 329–34
 - Document Object Model (DOM). *See also*
 - Legacy DOM, defined; W3C DOM (Document Object Model)
 - browser support for, 13
 - compared with Browser Object Model (BOM), 271–72
 - defined, 13, 613
 - and JavaScript objects, 13
 - Level 1, 13, 14
 - Level 2, 13, 14
 - overview, 13–15
 - role of W3C, 13
 - tree structure, 13–14, 15
 - document* property
 - frame* object, 308
 - window* object, 285
 - DOM. *See* Document Object Model (DOM);
 - Legacy DOM, defined; W3C DOM (Document Object Model)
 - DOM inspectors, 621–22
 - domain* property, *document* object, 330
 - dot metacharacter, 736–37
 - dot notation, 13, 38, 176, 178, 180, 222–23, 328–29, 333, 353, 837
 - double quote marks ("), 54
 - double words, defined, 109
 - do/while* loop, 133–34
 - drop-down menus
 - dynamics, 601–6
 - programming, 385–92
 - dwords, defined, 109
 - Dynamic HTML (DHTML), 585
- ## E
- E* property, *Math* object, 241
 - Eclipse IDE, 29
 - ECMAScript, 12–13, 17
 - Eich, Brendan, 1, 17, 18
 - Element DOM object, defined, 613
 - element nodes, DOM, defined, 614
 - elements, form, programming, 372–401
 - elements* property, *forms* object, 349
 - e-mail addresses
 - preliminary HTML form validation, 405–7
 - using regular expressions in HTML form validation, 781–83
 - embedded style sheets, 550–53
 - enabledPlugin* property, *mime* object, 283
 - encoding* property, *forms* object, 349
 - equal sign (=)
 - as assignment operator, 84, 90–91
 - in comparison operators, 95–98
 - equal-to (==) operator, 41, 58, 96–97
 - identical-to (===) operator, 96, 97
 - not-equal-to (!=) operator, 95
 - not-identical-to (!==) operator, 96, 97
 - and operator precedence, 84, 85
 - equality operators. *See* comparison operators
 - errors, types of, 40–41. *See also* debugging
 - escape()* built-in function, 699–700, 702–4
 - escape sequences, for strings, 54–56
 - European Computer Manufacturers Association (ECMA), 12–13
 - eval()* function, 118–19, 843–48
 - event handlers
 - as attributes of HTML tags, 456–58, 459
 - compared with JavaScript event methods, 462–63, 496, 497–99
 - comparing inline model and scripting model, 517, 661–62

event handlers (*Continued*)

- creating rollovers, 432, 476–78
 - for form events, 482–96
 - inline model, 455–65, 661
 - for link events, 481–82
 - list, with uses, 458–59
 - overview, 455
 - registering events, 456–57, 459, 460, 670–73
 - return values, 461–62
 - role in using JavaScript to submit forms, 359–65
 - scripting model, 455, 517–23, 661–62
 - setting up, 459–61
 - syntax, 456–57
 - triggered by mouse, 474–81
 - W3C DOM model, 662–81
- event handling, defined, 456
- event listeners
- adding, 668–70
 - Internet Explorer registration model, 676–78
 - multiple, adding, 670–73
 - removing, 673–76
- event methods, compared with HTML event handlers, 462–65, 496, 497–99
- event* object
- browser differences, 500–503
 - overview, 499–500
 - properties, Firefox, 502–3
 - properties, Internet Explorer, 501–2
- eventPhase* property, *event* object, 502, 679
- events
- affecting windows and frames, 465–74
 - bubbling and capturing, 500, 662–68
 - calling functions from, 149–51
 - defined, 359–60
 - DOM event properties, 678–82
 - Firefox *event* object properties, 502–3
 - Internet Explorer *event* object properties, 501–2
 - multiple, registering, 670–73
 - passing to JavaScript functions, 505–8
 - registering, 456–57, 459, 460
 - relationship to HTML, 11

- role of JavaScript, 10–12, 455
- simulating by applying methods to objects, 462–65

timed, creating, 292–303

exception handling, 168–72

exec() method, *RegExp* object, 723–24

exp() method, *Math* object, 242

expressions, defined, 83

external files

- importing CSS files, 557–58
- linking style sheets, 550, 555–58
- storing scripts in, 22, 144, 151

F

fgcolor HTML *<body>* tag attribute, 329

fgColor property, *document* object, 330

file, HTML *<form>* tag element, 337

filename property, *plugin* object, 279

files, external

- importing CSS files, 557–58
- linking style sheets, 550, 555–58
- storing scripts in, 22, 144, 151

FileUpload, HTML *<form>* tag element, 353

FileUpload object, event handlers, 483

FileUpload property, *forms* object, 349

finally clause, 170–72

Firebug (browser extension)

- debugging Ajax with, 848–52
- overview, 43–44

Firefox

- cookies, 697
 - debugging tools, 41–44, 166
 - displaying properties of *navigator* object, 274
 - event handling, 499–500, 501, 502–3, 504, 505, 506, 508, 510, 511, 514, 516
 - Firebug extension, 43–44, 848–52
 - invoking error console, 41–42
 - JavaScript in, 1–2, 18
 - Live Headers add on, 804
 - managing plug-ins, 279–80
 - support for DOM specification, 13, 14, 621, 622
 - Web site for, 15
- firstChild* DOM property, 614, 618

- fixed()* method, *String* object, 252
- float* CSS property, 534
- floor()* method, *Math* object, 242, 244
- focus()* event method, 463, 470–71, 496
- focus()* method
 - checkbox* object, 397
 - document* object, 334
 - frame* object, 308
 - password* object, 379
 - radio* object, 393
 - select* object, 386
 - text* object, 374
 - textarea* object, 382
 - window* object, 286
- font* CSS property, 533, 539
- fontcolor()* method, *String* object, 252
- font-family* CSS property, 533, 539
- fonts
 - list of common CSS properties, 533
 - specifying in Cascading Style Sheets, 539–41
 - units of measurement, 535–36
- font-size* CSS property, 533, 539
- fontsize()* method, *String* object, 252
- font-size-adjust* CSS property, 533
- font-stretch* CSS property, 533
- font-style* CSS property, 533, 539
- font-variant* CSS property, 533, 539
- font-weight* CSS property, 533, 539
- for* loop
 - as basic JavaScript looping construct, 134–35
 - populating arrays by using, 217–18
- for/in* loop, 135, 194–96
- form events, 482–96
- form object, in *forms []* array, 345, 353, 482–83. *See also forms* object
- form* property
 - checkbox* object, 396
 - elements* object, 350
 - password* object, 378
 - radio* object, 393
 - select* object, 385
 - text* object, 373
 - textarea* object, 382
- <form>* tag, HTML
 - associating events with, 456–57
 - attributes, 334–36
 - document example, 338–41
 - elements and properties, 353–56
 - input types, 337–38
 - and *onSubmit* event handler, 362–63
 - overview, 334
 - relationship of JavaScript *forms* object to, 341–42
- forms*. *See also* validating HTML forms
 - as HTML documents, 334–41
 - in JavaScript hierarchy, 328, 341–42
 - list of event handlers, 484
 - programming of controls, 372–401
 - relationship between JavaScript and HTML, 341–42
 - this* keyword for, 484–85
- forms []* array, 345–46, 347, 348, 353, 482–83
- forms* object
 - in JavaScript hierarchy, 328, 342
 - methods, 348, 349–50
 - overview, 342
 - properties, 348–49
 - as property of *document* object, 345–46, 348
 - relationship to HTML *<form>* tag, 341–42
- forms* property, as element of *document* object, 482–83
- forward()* method, *history* object, 319
- forward slash (/)
 - as division operator, 88
 - in regular expressions, 717, 720
- frame* object, 307–8
- frames
 - collapsing, 312–14
 - collapsing menus in, 312–14
 - creating in HTML, 304–7
 - creating menus in, 308–10, 314
 - creating navigation bars in, 308–10
 - dividing windows into, 303–14
 - handling events, 465–74
 - overview, 303
 - role of *location* object, 315–18

frames property

frame object, 308

window object, 285, 307

`<frameset>` tag, HTML, 303, 304

fromCharCode() method, *String* object, 253

fromElement property, *event* object, 501, 503, 679

Function() constructor, 264

Function object, 246, 264–66

function operator, 120

functions

 anonymous, as variables, 156–58

 assigning to properties, 185–87

 calling, 144–46

 calling from events, 149–51

 calling from JavaScript, 144–46, 151

 calling from links, 148–49

 as closures, 158–61

 compared with methods, 143, 180

 curly braces in, 144, 145

 debugging techniques, 166–72

 declaring, 144–46

 defined, 143

 inline, as methods, 185–87

 inner compared with outer, 158–61

 invoking, 144–46

 as JavaScript classes, 182

 as objects, 185–87

 overview, 143

 parentheses in, 144, 146, 156, 518

 passing arguments, 146–48

 returning values, 153–55

 scope of variables, 151–53

 storing definitions, 151

 syntax rules, 166

G

Garrett, Jesse James, 5

Gecko-based browsers, 13, 14

GET method, 335, 336, 803–5, 827–32

getAllResponseHeaders() method,

XMLHttpRequest object, 802, 810

getAttribute() DOM method, 630

getDate method, *Date* object, 236

getDay method, *Date* object, 236

getElementById() method, *document* object, 334, 350–52, 622–25, 658

getElementByName() method, *document* object, 334

getElementByTagName() method, *document* object, 334, 625–27

getFullYear method, *Date* object, 236

getHours method, *Date* object, 236

getMilliseconds method, *Date* object, 236

getMinutes method, *Date* object, 236

getMonth method, *Date* object, 236

getResponseHeader() method,

XMLHttpRequest object, 802, 810

getSeconds method, *Date* object, 236

getTime method, *Date* object, 236

getTimeZoneOffset method, *Date* object, 236

getUTCDate() method, *Date* object, 236

getUTCDay() method, *Date* object, 236

getUTCFullYear() method, *Date* object, 236

getUTCHours() method, *Date* object, 236

getUTCMilliseconds() method, *Date* object, 236

global property, *RegExp* object, 725

global variable scope, 66

go() method, *history* object, 319

Google Chrome, 15, 17, 834

Google Maps, 6–7, 797

Google Suggest, 6, 797–98, 799

grouping, in CSS structure, 531–32

Gustafson, Aaron, 683

H

handleEvent() method

checkbox object, 397

password object, 379

radio object, 393

select object, 386

text object, 374

textarea object, 382

hasAttributes() DOM method, 630

hasChildNodes() DOM method, 615, 630

hash property, *location* object, 315

hasOwnProperty() object method, 205

<head> tag, HTML
 declaring functions in, 144, 145
 in DOM tree-structure, 613, 614
 and placement of JavaScript code, 35
height CSS property, 534, 573
height property
 event object, 503
 image object, 417
 screen object, 323
 hexadecimal color codes, 537
 hexadecimal number system, 109
hidden, HTML <form> tag element
 attributes, 337
 description, 337, 353
 properties, 353
hidden object, event handlers, 483
hidden property, *forms* object, 349
history object
 JavaScript code example, 319–22
 methods, 319
 properties, 319
history property, *window* object, 285. *See also*
 history object
host property
 links object, 419
 location object, 315
hostname property
 links object, 419
 location object, 315
href property
 links object, 419
 location object, 315, 316
hreflang property, *links* object, 419
hsh property, *links* object, 419
hspace property, *image* object, 417
 HTML documents. *See also* <body> tag,
 HTML; HTML forms
 adding JavaScript to pages, 20, 21, 22–23,
 29–30
 creating, 29
 forms in, 334–41
 IDEs for, 29
 relationship of JavaScript to, 1–2, 10
 role of *document* object in, 329–34
 as static, 10

 validating markup, 24–25
 HTML forms
 checking alphabetic data input, 403–5,
 769–71
 checking credit card number input, 783–90
 checking e-mail address input, 405–7,
 781–83
 checking for empty fields, 401–3, 765–67
 checking for extraneous characters, 771–75
 checking password entries, 407–9
 checking phone number input, 777–80
 checking Social Security number input,
 775–77
 checking zip code input, 767–69
 fillout, submitting, 356–65
 JavaScript form event implementation,
 359–65
 list of controls, 337–38
 naming, 342–45, 350–56
 overview, 334
 regular expressions in validation, 765–94
 relationship of *form*> tag to JavaScript
 forms object, 341–42
 simple example, 338–41
 simple validation, 401–9
 <html> tag, HTML
 in DOM tree-structure, 613, 614
 HTTP response headers, 810
 HTTP status codes, 807–8
 hyphen (-). *See* minus sign (-)

I

id attribute, CSS, 565–66
id attribute, HTML form controls, 337–38,
 342, 517, 627
id property
 checkbox object, 396
 links object, 419
 password object, 378
 radio object, 393
 select object, 385
 text object, 373
 textarea object, 382
 IDEs (integrated development
 environments), 29, 30

- if* statements, 101
 - if/else if* statements, 127–28
 - if/else* statements, 108, 124–27
 - ignoreCase* property, *RegExp* object, 725
 - image*, HTML *<form>* tag element, 338, 358–59
 - Image()* constructor, 432–34
 - image maps
 - caching images, 432–34
 - creating, 436–38
 - displaying images randomly, 434–38
 - example, 425–28
 - overview, 423–25
 - shape coordinates, 425
 - using *src* property to replace images dynamically, 428–31
 - image* object
 - JavaScript hierarchy, 417
 - properties, 417
 - as property of *document* object, 416–17
 - using *src* property to replace images dynamically, 428–31
 - images
 - caching, 432–34
 - in Cascading Style Sheets, 533, 544–46
 - changing stick figure height and width properties in animation, 451–52
 - creating rollovers, 432, 476–78
 - creating slideshows by using controls, 442–45
 - creating slideshows by using mouse events, 478–81
 - displaying randomly, 434–38
 - HTML tags, 414
 - making clickable in slideshows, 445–48
 - overview, 413
 - preloading, 432–34
 - randomly displaying, 434–36
 - resizing to fit windows, 438–41
 - using in Web pages, 415–16
 - * tag, HTML, 414, 416, 417
 - importing CSS files, 557–58
 - in* operator, 120
 - indexOf()* method, *String* object, 253, 254
 - inheritance
 - creating subclasses, 207–9
 - implementing in JavaScript by using prototypes, 196–97
 - initEvent()* event method, 665
 - initializing variables, 60–62
 - inline functions, as methods, 185–87
 - inline model, event handling, 455
 - inline style sheets, 550, 553–54
 - innerHTML* property, 630–34
 - input devices, form, programming, 372–401
 - input* property, *RegExp* object, 725
 - insertBefore()* DOM method, 615, 630, 636–37
 - insertChild()* DOM method, 644
 - instance properties, *RegExp* object, 724, 725
 - instanceof* operator, 120, 205–6
 - integrated development environments (IDEs), 29, 30
 - Internet Explorer
 - and Browser Wars, 12
 - cookies, 697, 698
 - debugging tools, 44–46, 167
 - displaying properties of *navigator* object, 275
 - event handling, 499–500, 501–2, 503, 504, 505, 508, 510, 511, 512, 513, 516
 - event listener registration model, 676–78
 - JavaScript in, 1–2, 18, 19
 - and JScript, 13
 - managing plug-ins, 279
 - support for DOM specification, 13, 14, 621
 - testing whether JavaScript enabled, 18, 19
 - using Developer Tools, 44–45
 - Web site for, 15
 - isPrototypeOf()* object method, 205
 - italics()* method, *String* object, 252
- ## J
- Java, compared with JavaScript, 2–3
 - JavaScript
 - available libraries, 689–90
 - basic program example, 7–8
 - calling functions from, 144–46, 151
 - client-side compared to server-side, 2

- compared with Java, 2–3
- current state, 17, 18
- debugging tools, 41–46
- defined, 1
- as dynamic, 2, 10
- embedding code in HTML documents, 29–30
- enclosing in comment tags, 47–49
- enclosing in `<noscript>` tags, 49–50
- example of dynamic Web page, 2
- executing scripts in browser windows, 30–33
- history, 16–17
- latest version, 3
- as loosely typed language, 62–65
- overview, 1–3
- placement in HTML documents, 20–21
- relationship to ECMAScript, 12–13
- relationship to HTML, 1–2, 10, 29–33
- relationship to JScript, 13
- reserved keywords, 34
- role in Web page, 4–5
- role of `<script>` tag, 35–37
- statements and comments in, 34–35
- syntax and rules, 1, 33–37
- testing version in use, 17–18
- testing whether enabled, 18–19
- types of errors, 40–41
- as unobtrusive, 10, 682–89
- versions, 3, 16–17
- viewing output in browser, 37–40
- when to keep separate from HTML documents, 22–23
- `join()` method, arrays, 227
- .js files, 22, 144, 151
- JScript
 - relationship to JavaScript, 13
 - versions, 16–17
- JSON (JavaScript Object Notation)
 - browser support, 843–48
 - data structures, 835–36
 - examples, 839–43
 - overview, 834–35
 - steps to using, 836–39

K

- key events, 513–16
- `keyCode` property, *event* object, 501, 513
- keys, in arrays, 221, 225–27
- Komodo Edit, 29
- Konqueror browser, 13, 14, 15

L

- labels, for control statements in loops, 139–40
- `lang` property, as HTML attribute, 627
- `language` HTML `<script>` tag attribute, 36
- `lastChild` DOM property, 614, 618
- `lastIndex` property, *RegExp* object, 721, 725, 726–27
- `lastIndexOf()` method, *String* object, 253, 254
- `lastMatch` property, *RegExp* object, 725
- `lastModified` property, *document* object, 330
- `lastParen` property, *RegExp* object, 725
- layers, Web page, 682–89
- `layerX` property, *event* object, 502
- `layerY` property, *event* object, 502
- leaf nodes, DOM, defined, 614
- `left` property, in positioning CSS elements, 573, 574
- left shift (`<<`) operator, 110–12
- `leftContext` property, *RegExp* object, 725
- Legacy DOM, defined, 327, 342, 345. *See also document* object
- `length` property
 - arrays, 219, 220
 - forms* object, 349
 - frame* object, 308
 - Function* object, 264
 - history* object, 319
 - plugin* object, 279
 - select* object, 385
 - window* object, 285
- `letter-spacing` CSS property, 533, 542
- life cycle, Web page, 4–5
- `line-height` CSS property, 533, 542
- link events, 481–82
- `link` HTML `<body>` tag attribute, 329
- `link()` method, *String* object, 252
- `link` object, as property of *document* object, 418. *See also links* object

<link> tag, HTML, 555–56
linkColor property, *document* object, 330
 links
 assigning slideshow images to, 446–48
 associating with image maps, 436–38
 creating rollovers, 476–78
 overview, 417–18
links object
 example, 419–23
 in JavaScript hierarchy, 418
 overview, 418
 properties, 420
 LiveScript, 1
LN2 property, *Math* object, 241
LN10 property, *Math* object, 241
 load-time errors, 40
 local variable scope, 66
location object
 example, 316–18
 methods, 316
 overview, 315
 properties, 315
 syntax, 315
location property. *See also* *location* object
 document object, 330
 window object, 285
log() method, *Math* object, 242, 244
LOG2E property, *Math* object, 242
Log10E property, *Math* object, 242
 logical AND operator (&&), 101–3
 logical errors, 40–41
 logical NOT operator (!), 101, 105–6
 logical operators, 101–7
 logical OR operator(||), 101, 103–5
 loops
 breaking out of, 136–37
 controlling, 136–37
 defined, 131
 do/while loop, 133–34
 for/in loop, 135
 labeling control statements, 139–40
 for loop, 134–35
 nested, 137–38
 while loop, 131–33
lowsrc property, *image* object, 417

M

<map> tag, 414
margin CSS property, 534, 548
margin-bottom CSS property, 534, 548
margin-left CSS property, 534, 548
margin-right CSS property, 535, 548
margin-top CSS property, 535, 548
 margins, in Cascading Style Sheets, 534–35, 547–49
 markup, for Web documents, 9, 24–25
match() method, *String* object, 258, 727–28
Math object
 examples, 243, 245, 246
 generating random numbers, 245–46
 methods, 242–43
 overview, 241
 properties, 241–42
 rounding numbers up or down, 244–45
max() method, *Math* object, 242
maxLength property, *password* object, 378
MAX_VALUE property, *Number* object, 260, 261
media property, *links* object, 419
 menus
 collapsing in frames, 312–14
 creating in frames, 308–10, 314
 drop-down, 385–92, 601–6
 metacharacters. *See also* regular expressions
 alternative patterns, 759–65
 anchoring, 754–59
 and character class, 738–41
 compared with metasymbols, 741
 defined, 717, 733
 dot metacharacter, 736–37
 "greed" factor, 745–54
 overview, 733
 quantifiers, 745–54
 table of characters, 734–36
metaKey property, *event* object, 502
 metasymbols
 compared with metacharacters, 741
 defined, 733, 741
 examples, 742–45
 table of symbols, 742
method attribute, <*form*> tag, 335–36, 356, 826, 827

method property, *forms* object, 349

methods

adding by using prototypes, 202–4

applying to objects to simulate events,
462–65

for arrays, 227–33

checkboxes, 397

compared with functions, 143, 180

compared with properties, 333

Date object, 235–38

defined, 176

document object, 333–34

forms object, 348, 349–50

history object, 319

inline functions as, 185–87

location object, 316

Math object, 242–43

nodes, 615

Number object, 260

overview, 180–82

password object, 379

radio object, 393

RegExp object, 721–24

select object, 386

String object, 251–59, 727–33

text object, 374

textarea object, 382

window object, 286–87

XMLHttpRequest object, 801, 802

Microsoft Internet Explorer. *See* Internet Explorer

MIME (Multipurpose Internet mail

extensions) types, 282–84

mimeType object, 282–84

*mimeType*s property, *navigator* object, 273,
283

min() method, *Math* object, 242

minus sign (-)

auto-decrement (--) operator, 91–94

bitwise NOT (~) operator), 110, 111–12

as subtraction operator, 88

MIN_VALUE property, *Number* object, 260, 261

modifiers property, *event* object, 503

mouse events

creating rollovers, 476–78

creating slideshows, 478–81

how to use, 475–76

list of event handlers, 474

moveBy() method, *window* object, 291

moveTo() method, *window* object, 291

moving windows, 291–92

Mozilla Firefox. *See* Firefox

multiline property, *RegExp* object, 725

multiple property, *select* object, 385

N

name attribute, HTML form controls, 337–38,
342, 350, 358

name property

checkbox object, 396

elements object, 350

forms object, 349

frame object, 308

image object, 417

links object, 419

password object, 378

plugin object, 279

radio object, 393

select object, 385

text object, 373

textarea object, 382

window object, 285

naming variables, 60

NaN property, *Number* object, 260, 261–62

navigation bars, creating in frames, 308–10

navigator object

detecting browser, 276–78

detecting plug-ins, 278–82

JavaScript code example, 273–75

properties, 273

NEGATIVE_INFINITY property, *Number*
object, 260, 261

nested arrays, 223–27

nested loops, 137–38

NetBeans, 29

Netscape, 1, 2, 12, 573

new operator

defined, 120

role in constructors, 177–78, 182

role in creating *Array* objects, 214–16

- next* property, *history* object, 319
 - nextSibling* DOM property, 614, 618
 - Node DOM object
 - defined, 613
 - overview, 613–17
 - nodeName* DOM property, 615, 616
 - nodes
 - cloning, 648–53
 - list of methods, 615
 - list of properties, 614–15
 - overview, 613–14
 - parents and children, 615–16
 - removing, 653–58
 - siblings, 616
 - nodeType* DOM property, 615, 616
 - nodeValue* DOM property, 615
 - <*noscript*> tag, HTML, 49–50
 - NOT operators
 - bitwise (-), 110, 111–12
 - logical (!), 101, 105–6
 - null* keyword, 58–59
 - Number()* function, 113, 261
 - Number* object
 - JavaScript code examples, 261–63
 - methods, 260
 - overview, 259–60
 - properties, 260
 - as wrapper object, 246, 247
 - numeric arrays, 213
 - numeric data type, 53–54, 63, 112–19
- O**
- Object()* constructor, 177–78
 - object literals, 187–91
 - Object* object, 176, 178, 200, 204
 - objects. *See also* Browser Object Model (BOM); core objects; Document Object Model (DOM)
 - creating with constructors, 177–78
 - extending with prototypes, 196–209
 - hierarchical tree-like structure, 176
 - manipulating by using *for/in* loop, 194–96
 - manipulating by using *with* keyword, 191–94
 - methods overview, 176, 180–82
 - overview, 175–76
 - properties overview, 175, 178–79
 - types, 176
 - user-defined compared with built-in, 204–5
 - offscreenBuffering* property, *window* object, 285
 - offsetX* property, *event* object, 501
 - offsetY* property, *event* object, 501
 - older browsers, and JavaScript, 47–50
 - onAbort* event handler, 12, 458
 - onBlur* event handler, 12, 458, 465, 468–70, 484, 487–89
 - onblur* scripting model event handler property, 519
 - onChange* event handler, 12, 458, 484, 489–90
 - onchange* scripting model event handler property, 519
 - onClick* event handler
 - associating with image map links, 436–38
 - as attribute of HTML tag, 360–61, 459, 661
 - defined, 458, 474, 481
 - example, 460–61
 - and forms, 486–87
 - in inline model for handling events, 455, 456
 - overview, 11, 12, 484, 496
 - randomly displaying images, 434–36
 - registering, 459–60
 - syntax example, 456–57
 - uses for, 458
 - in *wakeUpCall()* function example, 457–58
 - onclick* scripting model event handler property, 517, 519
 - onDbClick* event handler, 458, 474
 - ondblclick* scripting model event handler property, 519
 - onDragDrop* event handler, 458
 - onError* event handler, 12, 458, 498–99
 - onerror XMLHttpRequest* event handler property, 802
 - onFocus* event handler, 12, 458, 465, 468–70, 484, 487–89

- onfocus* scripting model event handler
 - property, 519
- onKeyDown* event handler, 458, 513
- onkeydown* scripting model event handler
 - property, 519
- onKeyPress* event handler, 458, 513, 516
- onkeypress* scripting model event handler
 - property, 519
- onKeyUp* event handler, 458, 513
- onkeyup* scripting model event handler
 - property, 519
- onLoad* event handler, 12, 438, 458, 465–67, 496
- onload* scripting model event handler
 - property, 518, 519, 661
- onload XMLHttpRequest* event handler
 - property, 802
- onMouseDown* event handler, 474
- onmousedown* scripting model event handler
 - property, 519
- onMouseMove* event handler, 474, 475
- onmousemove* scripting model event handler
 - property, 519
- onMouseOut* event handler, 12, 432–34, 458, 474, 475, 476, 481, 521
- onMouseOver* event handler, 12, 432–34, 459, 474, 475, 476, 481, 521
- onmouseover* scripting model event handler
 - property, 517, 519, 521
- onMouseUp* event handler, 474
- onmouseup* scripting model event handler
 - property, 519
- onMove* event handler, 459, 465
- onprogress XMLHttpRequest* event handler
 - property, 802
- onreadystatechange XMLHttpRequest* event handler
 - property, 802, 806, 808
- onReset* event handler, 363–65, 459, 484
- onResize* event handler, 459, 472–74
- onSelect* event handler, 459, 484
- onSubmit* event handler
 - as attribute of `<form>` tag, 362–63
 - defined, 12, 459, 484
 - and forms, 491–96
 - in inline model for handling events, 455, 456
- onsubmit* scripting model event handler
 - property, 519
- onUnload* event handler, 12, 459, 465–67
- onunload* scripting model event handler
 - property, 519
- open()* method
 - document* object, 334
 - window* object, 286, 287, 289
 - XMLHttpRequest* object, 802, 803
- opener* property, *window* object, 286
- opening windows, 287–90
- Opera browser
 - cookies, 698
 - displaying properties of *navigator* object, 274
 - DOM inspector, 621
 - event handling, 501, 505, 507, 510, 513, 514
 - JavaScript error console, 168
 - JavaScript in, 18
 - support for DOM specification, 14
 - Web site for, 15
- operands
 - comparing, 95–100
 - defined, 83
- operators
 - arithmetic, 88–89
 - associativity, 84–88
 - auto-decrement, 91–94
 - auto-increment, 91–94
 - bitwise, 109–12
 - comparison, 95–100
 - conditional, 108–9
 - defined, 83
 - logical, 101–7
 - order of evaluation, 84–88
 - precedence, 84–88
 - shortcut assignment, 90–91
- options[]* property, *select* object, 385
- OR operators
 - bitwise (`|`), 110, 111–12
 - logical (`||`), 101, 103–5
- overflow* property, in positioning CSS elements, 573
- ownerDocument* DOM property, 615

- P**
- padding* CSS property, 535, 548
 - padding-bottom* CSS property, 535, 548
 - padding-left* CSS property, 535, 548
 - padding-right* CSS property, 535, 548
 - padding-top* CSS property, 535, 548
 - pageX* property, *event* object, 502, 508
 - pageY* property, *event* object, 502, 508
 - parent* property
 - frame* object, 308
 - window* object, 286
 - parentheses ()
 - following method names, 13, 38, 180, 333
 - in function statements, 144, 146, 156, 518
 - and operator precedence, 84, 85
 - and regular expressions, 736, 761, 762, 772–73, 777–80, 781, 790
 - parentNode* DOM property, 615, 618
 - parse()* method, *Date* object, 236
 - parseFloat()* function, 116–18
 - parseInt()* function, 114–16
 - password*, HTML *<form>* tag element
 - attributes, 337
 - description, 337, 353
 - properties, 353
 - password* object
 - event handlers, 483
 - JavaScript code example, 379–81
 - JavaScript hierarchy, 378
 - methods, 379
 - overview, 377
 - properties, 378
 - password* property, *forms* object, 349
 - passwords, HTML form validation, 407–9
 - pathname* property
 - links* object, 419
 - location* object, 315
 - Perl, similarity to JavaScript, 1
 - phone numbers, validating, 777–80
 - PI* property, *Math* object, 242, 243–44
 - pixelDepth* property, *screen* object, 323
 - platform* property, *navigator* object, 273
 - plugin* object, 279–80
 - plug-ins, detecting, 278–82
 - plus sign (+)
 - as addition operator, 88
 - auto-increment (++) operator, 91–94
 - as concatenation operator, 66, 94–95
 - poly* shape, image map, 425
 - pop()* method, arrays, 227, 229
 - popup windows, 287, 288, 370–72
 - port* property
 - links* object, 419
 - location* object, 315
 - position* property, in positioning CSS elements, 573
 - POSITIVE_INFINITY* property, *Number* object, 260, 261
 - POST method, 335, 336, 803–5, 833–34
 - pow()* method, *Math* object, 242, 243–44
 - precedence, operator, 84–88
 - preventDefault()* event method, 665
 - previous* property, *history* object, 319
 - previousSibling* DOM property, 615, 618
 - primitive data types
 - boolean, 56–57
 - numeric, 53–54
 - overview, 53
 - string, 54–56
 - print()* method, *frame* object, 308
 - progressive enhancement, 683
 - prompt()* method, *window* object, 76–78, 286
 - properties
 - applying to regular expressions, 724–27, 724–27
 - arrays, 219–21
 - assigning functions to, 185–87
 - background, 178–79
 - buttons, 353
 - Cascading Style Sheets, 532–49
 - checkboxes, 353, 396
 - defined, 175
 - document* object, 330
 - DOM events, 678–82
 - event* object, 501–3
 - <form>* tag elements, 353–56
 - forms* object, 348–49
 - frame* object, 308
 - history* object, 319
 - image* object, 417

- links* object, 419
- location* object, 315
- lookup chain, 199–202
- Math* object, 241–42
- navigator* object, 273–76
- nodes, 614–15
- Number* object, 260
- passwords, 353, 378
- plugin* object, 279
- radio buttons, 353, 393
- RegExp* object, 724–27
- screen* object, 322–23
- select* object, 385
- String* object, 249–50
- text* object, 373–74
- textarea* object, 382
- window* object, 285–86
- XMLHttpRequest* object, 801, 802
- protocol* property
 - links* object, 420
 - location* object, 315
- prototype lookup chain, 199–202
- prototype* object, 196–97
- prototype* property
 - Array* object, 219, 220
 - Boolean* object, 264
 - customizing *Date* objects, 240–41
 - defined, 196, 204, 240
 - Function* object, 264
 - image* object, 417
 - Number* object, 260
 - purpose, 196, 197
 - in subclasses, 207–10
 - using to add properties to classes, 198–99
- prototypes
 - assigning methods, 202–4
 - assigning properties, 198–99
 - for extending objects, 196–209
 - implementing inheritance, 196–97
 - lookup chain, 199–202
 - overview, 196
- push()* method, arrays, 227, 229–30

Q

- quote marks, double ("), 54
- quote marks, single ('), 54

R

- radio*, HTML *<form>* tag element
 - attributes, 337
 - description, 337, 353
 - properties, 353
- radio* object
 - event handlers, 483
 - JavaScript code example, 394–95
 - in JavaScript hierarchy, 392
 - methods, 393
 - overview, 392–93
 - properties, 393
- radio* property, *forms* object, 349
- random()* method, *Math* object, 242, 245–46, 434–36
- random numbers, 245–46
- readOnly* property
 - password* object, 378
 - text* object, 374
 - textarea* object, 382
- readyState* property, *XMLHttpRequest* object, 802, 806–7
- reason* property, *event* object, 502
- rect* shape, image map, 425
- recursion, JavaScript support, 161–65
- referrer* property, *document* object, 330
- RegExp()* constructor, 177, 720. *See also*
 - regular expressions
- RegExp* object
 - methods, 721–24
 - overview, 720
 - properties, 724–27
 - syntax, 720
- regular expressions
 - alternative patterns in, 759–65
 - applying properties, 724–27
 - backslash (\) and, 733, 734, 735
 - capturing subpatterns of characters, 762–65
 - comparing Perl and JavaScript, 717
 - creating by using constructor method, 177, 720

- regular expressions (*Continued*)
 - creating object with literal notation, 719–20
 - defined, 717
 - forward slash (/) and, 717, 720
 - grouping characters, 761–62
 - metacharacters in, 733–65
 - metasymbols in, 733, 741–45
 - overview, 717–19
 - String* object methods for, 727–33
 - testing, 721–24
 - in validating forms, 765–94
 - rel* property, *links* object, 419
 - relatedTarget* property, *event* object, 502
 - relative positioning, CSS elements, 581–83
 - reload()* method, *location* object, 316
 - removeChild()* DOM method, 615, 630, 653–58
 - removeEventListener()* method, 673
 - replace()* method
 - form validation examples, 771–75
 - location* object, 316
 - String* object, 253, 258–59, 730–31, 771–75
 - replaceChild()* DOM method, 615, 630
 - reset*, HTML *<form>* tag element
 - attributes, 338
 - description, 338, 353
 - properties, 353
 - reset()* event method, 463
 - reset()* method, *forms* object, 349, 368–70
 - reset* object, event handlers, 483
 - reset* property, *forms* object, 349
 - resizeBy()* method, *window* object, 291
 - resizeTo()* method, *window* object, 291
 - resizing windows, 291–92
 - responseText* property, *XMLHttpRequest* object, 802, 808–9
 - responseXML* property, *XMLHttpRequest* object, 802, 809
 - return* statement, 153–55
 - returnValue* property, *event* object, 501
 - rev* property, *links* object, 419
 - reverse()* method, arrays, 227
 - right* property, in positioning CSS elements, 573
 - right shift (>>) operator, 110–12
 - rightContext* property, *RegExp* object, 725
 - rollovers
 - creating with mouse events, 476–78
 - scripting model example, 521–23
 - simple imagemap example, 432–34
 - root node, DOM, 614
 - round()* method, *Math* object, 242, 244
 - rounding numbers up and down, 244–45
 - rows* property, *textarea* object, 382
 - rules* array, Microsoft, 585
 - runtime errors, 40
- S**
- Safari, 13, 14, 15, 18, 275
 - screen* object
 - JavaScript code example, 323–25
 - properties, 322–23
 - screen* property, *window* object, 286. *See also* *screen* object
 - screenX* property, *event* object, 502
 - screenY* property, *event* object, 502
 - <script>* tag, HTML
 - attributes, 36–37
 - calling functions from, 144–45, 151
 - and placement of JavaScript code, 35–37
 - scripting model for event handling, 455, 517–23
 - scroll()* method, *window* object, 287
 - scrolling messages
 - in body of Web documents, 658–61
 - in windows, 296–303
 - scrollTo()* method, *window* object, 298–302
 - search()* method, *String* object, 254, 258–59, 729–30
 - search patterns, in regular expressions, 717, 720, 733. *See also* metacharacters
 - search* property
 - history* object, 319
 - links* object, 420
 - location* object, 315
 - select*, HTML *<form>* tag element
 - attributes, 337
 - description, 337, 353
 - properties, 353

- `select()` event method, 463
- `select()` method
 - password object, 379
 - text object, 374
 - textarea object, 382
- `select` object
 - event handlers, 483
 - JavaScript code examples, 386–90
 - JavaScript hierarchy, 386
 - methods, 386
 - multiple selects, 390–92
 - overview, 385
 - properties, 385
- `select` property, *forms* object, 349
- `selectedIndex` property, *select* object, 385, 390
- selectors, in style sheet rules
 - contextual, 569–72
 - defined, 529
 - examples, 529–30, 570–72
 - grouping, 531–32
 - and inheritance, 569–72
 - nested, 569–72
- `self` property
 - frame object, 308
 - window object, 286
- semicolons, in JavaScript, 34–35
- `send()` method, *XMLHttpRequest* object, 802, 805–6
- server-side JavaScript, 2
- `setAttribute()` DOM method, 637–39
- `setAttributeNode()` DOM method, 630
- `setDate()` method, *Date* object, 236
- `setFullYear()` method, *Date* object, 236
- `setHours()` method, *Date* object, 236
- `setInterval()` method
 - frame object, 308
 - syntax, 449
 - window object, 287, 293–94
- `setMilliseconds()` method, *Date* object, 236
- `setMinutes()` method, *Date* object, 236
- `setMonth()` method, *Date* object, 236
- `setRequestHeader()` method, *XMLHttpRequest* object, 802, 811
- `setSeconds()` method, *Date* object, 236
- `setTime()` method, *Date* object, 237
- `setTimeout()` method
 - frame object, 308
 - syntax, 449
 - window object, 287, 293–94
- `setUTCdate()` method, *Date* object, 237
- `setUTCFullYear()` method, *Date* object, 237
- `setUTCHours()` method, *Date* object, 237
- `setUTCMilliseconds()` method, *Date* object, 237
- `setUTCMinutes()` method, *Date* object, 237
- `setUTCMonth()` method, *Date* object, 237
- `setUTCSeconds()` method, *Date* object, 237
- `setYear()` method, *Date* object, 237
- `shift()` method, arrays, 227, 230–31
- shift operators, bitwise, 110–12
- `shiftKey` property, *event* object, 501, 502
- shortcut assignment operators, 90–91
- `sin()` method, *Math* object, 242
- single quote marks ('), 54
- size property
 - password object, 378
 - select* object, 385
 - text object, 374
- slash (/). *See also* backslash (\) and regular expressions
 - as division operator, 88
 - in regular expressions, 717, 720
- `slice()` method
 - arrays, 227, 231–32, 233
 - String* object, 254
- slideshows
 - creating with mouse events, 478–81
 - making images clickable, 445–48
 - overview, 441–42
 - simple, creating, 442–45
- `small()` method, *Font* object, 252
- sniffers, browser, 15–16, 276–78
- Social Security numbers, validating, 775–77
- `sort()` method, arrays, 227
- `source` property, *RegExp* object, 725
- space, between words in JavaScript, 33–34
- `` tag, role in CSS, 566–72
- `splice()` method
 - arrays, 227, 232–33
 - String* object, 257

split() method, *String* object, 254, 257, 258, 731–33

sqrt() method, *Math* object, 242, 243–44

SQRT1_2 property, *Math* object, 242

SQRT2 property, *Math* object, 242

square bracket [] notation, 213, 216, 222–23, 353

square root, 243

src HTML `<script>` tag attribute, 36

src property, *image* object, 417, 428–31

srcElement property, *event* object, 501, 503, 504, 679

srcFilter property, *event* object, 502

statements, in JavaScript, 34–35

status property

- window* object, 286
- XMLHttpRequest* object, 802

statusText property, *XMLHttpRequest* object, 802

stick figures, in animations, 449, 450–52

stopPropagation() event method, 665

strike() method, *String* object, 252

String() function, 113–14

String object

- extending, 250–51
- methods, 251–59, 727–33
- overview, 247
- properties, 249–50
- and regular expressions, 727–33
- as wrapper object, 246, 247

strings

- as array index values, 221–27
- concatenation, 37, 56, 66–67
- converting data type, 63, 112–19
- defined, 37
- enclosing in quotes, 54–56, 69
- escape sequences, 54–56

style, role in Web page design, 9–10

style classes

- applying class selectors, 562–64
- defining, 558–60
- table styling example, 560–62

style HTML `` tag attribute, 566, 567–68

style object

- JavaScript code example, 592–94

- overview, 589
- properties, 589–91

style property, as HTML attribute, 594–96, 627

style sheets. *See also* Cascading Style Sheets (CSS)

- defined, 527
- how they work, 529–30
- for HTML pages, 527–28

`<style>` tag, HTML, 550–53

styleSheets array (property of *document* object), 585–87

sub() method, *String* object, 252

subclasses, and inheritance, 207–10

submit, HTML `<form>` tag element

- attributes, 338
- description, 338, 353
- properties, 353

submit() event method, 463

submit() method, *forms* object, 341, 349, 368–70

submit object, event handlers, 483

submit property, *forms* object, 349

substr() method, *String* object, 254, 257, 258

substring() method, *String* object, 257

subtraction operator (-), 88

suffixes property, *mime* object, 283

Sun Microsystems, 2

sup() method, *String* object, 252

switch statement, 128–31

symbols. *See* metasymbols

T

tabIndex property

- checkbox* object, 396
- password* object, 378
- radio* object, 393
- select* object, 385
- text* object, 374
- textarea* object, 382

tables

- cloning, 652–53
- creating, 644–48

tan() method, *Math* object, 243

- target* property
 - event* object, 503, 504, 679
 - forms* object, 349
 - links* object, 420
- telephone numbers, validating, 777–80
- ternary operator, 108
- test()* method, *RegExp* object, 721–23
- text*, HTML `<form>` tag element
 - attributes, 337
 - description, 337, 353
 - properties, 353
- Text DOM object, defined, 613
- text nodes, DOM, defined, 614
- text* object
 - event handlers, 483
 - JavaScript code examples, 374–77
 - JavaScript hierarchy, 373
 - methods, 374
 - overview, 373
 - properties, 373–74
- text* property, *forms* object, 349
- text-align* CSS property, 534, 542
- textarea*, HTML `<form>` tag element
 - attributes, 337
 - description, 337, 353
 - properties, 353
- textarea* object
 - event handlers, 483
 - JavaScript code example, 383–84
 - JavaScript hierarchy, 383
 - methods, 382
 - overview, 381
 - properties, 382
- textarea* property, *forms* object, 349
- text-decoration* CSS property, 533, 542
- text-indent* CSS property, 534, 542
- text-transform* CSS property, 533, 542
- this* keyword
 - for buttons, 484–85
 - in class example, 182–85
 - defined, 120, 182
 - for forms, 365–68, 484–85
 - in W3C event handlers, 669–70
- this* operator. *See this* keyword
- throw* statement, 170
- time, basic units, 239–40, 293
- timed events, creating, 292–303
- timer methods, 292–93
- timeStamp* property, *event* object, 503, 679
- title* HTML `` tag attribute, 566
- title* property
 - document* object, 330
 - as HTML attribute, 627
- `<title>` tag, in DOM tree-structure, 613, 614
- toElement* property, *event* object, 501, 503
- toExponential()* method, *Number* object, 260, 262
- toFixed()* method, *Number* object, 260, 262–63
- toGMTString()* method, *Date* object, 237
- toLocaleLowerCase()* method, *String* object, 254
- toLocaleString()* method
 - arrays, 227
 - Date* object, 237
 - Number* object, 260
- toLocaleUpperCase()* method, *String* object, 254
- toLowerCase()* method, *String* object, 254
- tooltips, 601, 606–8
- top* property
 - frame* object, 308, 310–12
 - in positioning CSS elements, 573, 574
 - window* object, 286
- toPrecision()* method, *Number* object, 260
- toSource()* method, *Date* object, 237
- toString()* method
 - arrays, 227
 - Boolean* object, 263
 - Date* object, 237
 - defined, 204
 - Math* object, 243
 - Number* object, 260, 261
 - String* object, 254
- toUpperCase()* method, *String* object, 254
- toUTCString()* method, *Date* object, 237
- try/catch* statements, 168, 169–70, 800
- type* HTML `<script>` tag attribute, 36
- type* HTML `<style>` tag attribute, 550–52

type property
 checkbox object, 396
 elements object, 350
 event object, 502, 503, 504, 679
 mimeType object, 283
 password object, 378
 radio object, 393
 select object, 385
 text object, 374
 textarea object, 382
 typeof operator, 57–58, 144

U

undefined keyword, 58–59
 unescape() built-in function, 702–4
 unobtrusive JavaScript, 10, 682–89
 unshift() method, arrays, 227, 230–31
 unwatch() method
 checkbox object, 397
 frame object, 308
 location object, 316
 password object, 379
 radio object, 393
 select object, 386
 text object, 374
 textarea object, 382
 URL property, document object, 330
 userAgent property, navigator object, 273
 UTC() method, Date object, 237

V

validating HTML forms
 alphabetic data input, 403–5, 769–71
 checking for empty fields, 401–3, 765–67
 credit card number input, 783–90
 e-mail address input, 405–7, 781–83
 password input, 407–9
 phone number input, 777–80
 removing extraneous characters, 771–75
 Social Security number input, 775–77
 zip code input, 767–69
 validation tools, for Web page markup,
 24–25
 Validome validation tool for Web documents,
 25

value, as HTML attribute, 358
 value property
 checkbox object, 396
 elements object, 350
 password object, 378
 radio object, 393
 text object, 374
 textarea object, 382
 valueOf() method
 Date object, 237
 defined, 204
 String object, 254
 values, in array index keys, 221, 225–27
 var keyword, 60–62
 variables
 as anonymous functions, 156–58
 concatenating with strings, 66–67
 converting data type, 62, 63, 64–65
 declaring, 60–62, 69
 initializing, 60–62
 naming, 60
 overview, 59–60
 scope, local vs. global, 66
 scope in functions, 151–53
 vertical-align CSS property, 534, 542
 visibility property, in positioning CSS
 elements, 573, 601, 602, 606
 vlink HTML <body> tag attribute, 329
 vlinkColor property, document object, 330
 void operator, 120, 482
 vspace property, image object, 417

W

W3C DOM (Document Object Model)
 adding blog entries, 639–45
 cloning nodes, 648–53
 creating tables, 644–48
 event handling, 662–68
 HTML document node overview,
 612–13
 modifying, 629–61
 overview, 611
 removing nodes, 653–58
 scrolling marquee example, 658–61
 upside-down tree structure, 613–16

- `watch()` method
 - `checkbox` object, 397
 - `frame` object, 308
 - `location` object, 316
 - `password` object, 379
 - `radio` object, 393
 - `select` object, 386
 - `text` object, 374
 - `textarea` object, 382
 - Web browsers. *See* browsers
 - Web pages. *See also* HTML documents;
 - validating HTML forms
 - basic JavaScript program example, 7–8
 - examples of Ajax applications, 6–7, 797–98
 - how they work, 4–5
 - life cycle example, 4–5
 - role of JavaScript, 4–5, 8–10
 - validating markup, 24–25
 - `which` property, `event` object, 503, 510, 511
 - `while` loop, 131–33
 - whitespace
 - in DOM tree, 617
 - and metacharacters, 734, 742, 744
 - between words, 33–34
 - `width` property
 - CSS, 535, 560–61, 573
 - `event` object, 503
 - `image` object, 417
 - `screen` object, 323
 - `window` object. *See also* windows
 - methods, 286–87
 - overview, 285
 - properties, 285–86
 - as top-level browser object, 73, 179
 - `window` property
 - `frame` object, 308
 - `window` object, 286
 - windows
 - changing status bar, 295, 296
 - dividing into frames, 303–14
 - handling events, 465–74
 - moving and resizing, 291–92
 - opening and closing, 287–90
 - popup, 370–72
 - resizing images to fit, 438–41
 - role of `history` object, 319–22
 - role of `location` object, 315–18
 - scrolling messages, 296–303
 - with keyword
 - `String` object example, 266–67
 - and user-defined objects, 191–94
 - `word-spacing` CSS property, 534, 542
 - World Wide Web Consortium (W3C). *See*
 - also W3C DOM (Document Object Model)
 - browser standards, 12–13
 - `cssRules` array, 585
 - validation tool for Web documents, 24
 - wrapper objects
 - `Boolean` object, 246, 263–64
 - `Number` object, 246, 247, 259–63
 - `String` object, 246, 247–59
 - `write()` method, `document` object, 37, 38–39, 40, 331, 333, 334
 - `writeln()` method, `document` object, 37, 38–39, 40, 333, 334
- X**
- `x` property, `event` object, 502
 - `XMLHttpRequest` object
 - checking HTTP response headers, 810
 - creating, 800–802
 - handling server response with callback function, 808–10
 - initializing, 803–5
 - methods, 801, 802
 - monitoring server response, 806–8
 - properties, 801, 802
 - sending request to server, 805–6
 - XOR bitwise operator (^), 110, 111–12
- Y**
- `y` property, `event` object, 502
- Z**
- `z-index` property, in positioning CSS
 - elements, 573, 583–84, 596–98
 - zero-fill right shift (>>>) operator, 110–12
 - zip codes, validating, 767–69