

| A BEGINNER'S GUIDE

John
Pollock

Fourth Edition

JavaScript

- Create dynamic Web pages with special effects
- Learn the latest JavaScript standard
- Work with HTML5 and jQuery

JavaScript

A Beginner's Guide

Fourth Edition

John Pollock



New York Chicago San Francisco
Lisbon London Madrid Mexico City
Milan New Delhi San Juan
Seoul Singapore Sydney Toronto

Copyright © 2013 by The McGraw-Hill Companies. All rights reserved. Except as permitted under the United States Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the publisher, with the exception that the program listings may be entered, stored, and executed in a computer system, but they may not be reproduced for publication.

ISBN: 9780071809382

MHID: 0071809384

The material in this e-book also appears in the print version of this title: ISBN: 978-0-07-180937-5,

MHID: 0-07-180937-6

McGraw-Hill e-books are available at special quantity discounts to use as premiums and sales promotions, or for use in corporate training programs. To contact a representative please e-mail us at bulksales@mcgraw-hill.com.

All trademarks are trademarks of their respective owners. Rather than put a trademark symbol after every occurrence of a trademarked name, we use names in an editorial fashion only, and to the benefit of the trademark owner, with no intention of infringement of the trademark. Where such designations appear in this book, they have been printed with initial caps.

Information has been obtained by McGraw-Hill from sources believed to be reliable. However, because of the possibility of human or mechanical error by our sources, McGraw-Hill, or others, McGraw-Hill does not guarantee the accuracy, adequacy, or completeness of any information and is not responsible for any errors or omissions or the results obtained from the use of such information.

TERMS OF USE

This is a copyrighted work and McGraw-Hill and its licensors reserve all rights in and to the work. Use of this work is subject to these terms. Except as permitted under the Copyright Act of 1976 and the right to store and retrieve one copy of the work, you may not decompile, disassemble, reverse engineer, reproduce, modify, create derivative works based upon, transmit, distribute, disseminate, sell, publish or sublicense the work or any part of it without McGraw-Hill's prior consent. You may use the work for your own noncommercial and personal use; any other use of the work is strictly prohibited. Your right to use the work may be terminated if you fail to comply with these terms.

THE WORK IS PROVIDED "AS IS." THE MCGRAW-HILL COMPANIES AND ITS LICENSORS MAKE NO GUARANTEES OR WARRANTIES AS TO THE ACCURACY, ADEQUACY OR COMPLETENESS OF OR RESULTS TO BE OBTAINED FROM USING THE WORK, INCLUDING ANY INFORMATION THAT CAN BE ACCESSED THROUGH THE WORK VIA HYPERLINK OR OTHERWISE, AND EXPRESSLY DISCLAIM ANY WARRANTY, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. McGraw-Hill and its licensors do not warrant or guarantee that the functions contained in the work will meet your requirements or that its operation will be uninterrupted or error free. Neither McGraw-Hill nor its licensors shall be liable to you or anyone else for any inaccuracy, error or omission, regardless of cause, in the work or for any damages resulting therefrom. McGraw-Hill has no responsibility for the content of any information accessed through the work. Under no circumstances shall McGraw-Hill and/or its licensors be liable for any indirect, incidental, special, punitive, consequential or similar damages that result from the use of or inability to use the work, even if any of them has been advised of the possibility of such damages. This limitation of liability shall apply to any claim or cause whatsoever whether such claim or cause arises in contract, tort or otherwise.

*To my wife Heather, daughters Eva and Elizabeth,
Bruce and Joy Anderson, and Dr. J. D. and Linda Andrews*

*In memory of James D. and Livian Anderson, John William and Edith Hopkins,
Burley T. and Aline Price, "Doc" Flores, and Clifton Idom*

About the Author

John Pollock is employed as a Web Administrator during the day and works on Web sites and other projects during the evening. He runs a Web site devoted to Web development and design: Script the Web (www.scripttheweb.com). He also is a contributor to Web Xpertz (www.webxpertz.net), a help community for Web developers. John holds a Bachelor of Arts in English from Sam Houston State University and currently lives in Huntsville, Texas with his wife, Heather, and his daughters, Eva and Elizabeth.

About the Technical Editor

Christie Sorenson is a senior software engineer at ZingChart. She has worked on JavaScript-based systems in analytics, content management, and business applications since 1997 and has been fascinated with the evolution of the language and its users. She has collaborated on several books, including *Ajax: The Complete Reference* and *HTML & CSS: The Complete Reference* and was also the tech editor on *JavaScript: The Complete Reference* and *HTML: A Beginner's Guide*. She holds a Bachelor of Science in Computer Science from University of California, San Diego, and now lives in San Francisco with her husband, Luke, and daughters, Ali and Keira.

Contents at a Glance

1	Introduction to JavaScript	1
2	Placing JavaScript in an HTML File	15
3	Using Variables	33
4	Using Functions	59
5	JavaScript Operators	87
6	Conditional Statements and Loops	117
7	JavaScript Arrays	153
8	Objects	183
9	The Document Object	209
10	Event Handlers	243
11	Window Object	275
12	Math, Number, and Date Objects	305

13 Handling Strings	343
14 JavaScript and Forms	379
15 An Introduction to Advanced Techniques	411
16 JavaScript Libraries, HTML5, and Harmony	445
A Answers to Self Tests	477
Index	489

Contents

ACKNOWLEDGMENTS	xvii
INTRODUCTION	xix
1 Introduction to JavaScript	1
What You Need to Know	2
Basic HTML and CSS Knowledge	3
Basic Text Editor and Web Browser Knowledge	3
Which Version?	5
Remember, It's Not Java	6
Similarities to Other Languages	6
Beginning with JavaScript	8
Prototype-Based	8
Client-Side	8
Scripting Language	9
Putting It All Together	9
Online Resources	10
Try This 1-1: Use JavaScript to Write Text	10
Chapter 1 Self Test	11
2 Placing JavaScript in an HTML File	15
Using the HTML Script Tags	16
Identifying the Scripting Language	17
Calling External Scripts	17

Specifying When the Script Should Load	18
Using <noscript></noscript> Tags	18
Creating Your First Script	20
Writing a “Hello World” Script	20
Creating an HTML Document for the Script	21
Inserting the Script into the HTML Document	21
Try This 2-1: Insert a Script into an HTML Document	23
Using External JavaScript Files	24
Creating a JavaScript File	24
Creating the HTML Files	24
Viewing the Pages in Your Browser	26
Try This 2-2: Call an External Script from an HTML Document	27
Using JavaScript Comments	28
Inserting Comments on One Line	28
Adding Multiple-Line Comments	28
Chapter 2 Self Test	30
3 Using Variables	33
Understanding Variables	34
Why Variables Are Useful	35
Variables as Placeholders for Unknown Values	35
Variables as Time-Savers	35
Variables as Code Clarifiers	36
Defining Variables for Your Scripts	36
Declaring Variables	36
Assigning Values to Variables	37
Naming Variables	38
Understanding Data Types	41
Number	41
String	42
Boolean	46
Null	47
Undefined	47
Try This 3-1: Declare Variables	48
Using Variables in Scripts	49
Making a Call to a Variable	49
Adding Variables to Text Strings	50
Writing a Page of JavaScript	51
Creating the Framework	51
Defining the Variables	52
Adding the Commands	53
Modifying the Page	54
Try This 3-2: Create an HTML Page with JavaScript	55
Chapter 3 Self Test	57

4 Using Functions	59
What a Function Is	60
Why Functions Are Useful	60
Structuring Functions	61
Declaring Functions	61
Defining the Code for Functions	62
Naming Functions	63
Adding Arguments to Functions	63
Adding Return Statements to Functions	65
Calling Functions in Your Scripts	66
Script Tags: Head Section or Body Section	67
Calling a Function from Another Function	70
Calling Functions with Arguments	71
Calling Functions with Return Statements	75
Other Ways to Define Functions	75
Try This 4-1: Create an HTML Page with Functions	79
Scope/Context Basics	81
Global Context	81
Function Context	81
Try This 4-2: Write Your Own Functions	83
Chapter 4 Self Test	84
5 JavaScript Operators	87
Understanding the Operator Types	88
Understanding Arithmetic Operators	89
The Addition Operator (+)	90
The Subtraction Operator (−)	93
The Multiplication Operator (*)	93
The Division Operator (/)	94
The Modulus Operator (%)	95
The Increment Operator (++)	95
The Decrement Operator (−−)	96
The Unary Plus Operator (+)	96
The Unary Negation Operator (−)	97
Understanding Assignment Operators	98
The Assignment Operator (=)	99
The Add-and-Assign Operator (+=)	99
The Subtract-and-Assign Operator (−=)	100
The Multiply-and-Assign Operator (*=)	100
The Divide-and-Assign Operator (/=)	100
The Modulus-and-Assign Operator (%=)	100
Try This 5-1: Adjust a Variable Value	101
Understanding Comparison Operators	102
The Is-Equal-To Operator (==)	102
The Is-Not-Equal-To Operator (!=)	104

The Strict Is-Equal-To Operator (===)	105
The Strict Is-Not-Equal-To Operator (!==)	105
The Is-Greater-Than Operator (>)	106
The Is-Less-Than Operator (<)	106
The Is-Greater-Than-or-Equal-To Operator (>=)	107
The Is-Less-Than-or-Equal-To Operator (<=)	107
Understanding Logical Operators	108
The AND Operator (&&)	108
The OR Operator ()	108
The NOT Operator (!)	109
The Bitwise Operators	109
Special Operators	110
Understanding Order of Operations	111
Try This 5-2: True or False?	113
Chapter 5 Self Test	114
6 Conditional Statements and Loops	117
Defining Conditional Statements	118
What Is a Conditional Statement?	118
Why Conditional Statements Are Useful	119
Using Conditional Statements	119
Using if/else Statements	119
Using the switch Statement	127
Using the Conditional Operator	129
User Input from a Prompt	130
Try This 6-1: Work with User Input	133
Defining Loops	134
What Is a Loop?	134
Why Loops Are Useful	134
Using Loops	135
for	135
while	143
do while	143
for in and for each in	145
Using break and continue	145
Try This 6-2: Work with for Loops and while Loops	147
Chapter 6 Self Test	149
7 JavaScript Arrays	153
What Is an Array?	154
Why Arrays Are Useful	155
Defining and Accessing Arrays	155
Naming an Array	155
Defining an Array	155

Accessing an Array's Elements	157
Using the length Property and Loops	158
Changing Array Values and Changing the Length	159
Try This 7-1: Use Loops with Arrays	161
Array Properties and Methods	162
Properties	162
Methods	163
Nesting Arrays	176
Defining Nested Arrays	177
Loops and Nested Arrays	177
Try This 7-2: Nested Arrays Practice	180
Chapter 7 Self Test	181
8 Objects	183
Defining Objects	184
Creating Objects	184
Naming	184
Single Objects	185
Try This 8-1: Create a Computer Object	188
Object Structures	189
Constructor Functions	189
Using Prototypes	194
Helpful Statements for Objects	197
The for-in Loop	197
The with Statement	198
Try This 8-2: Practice with the Combination Constructor/Prototype Pattern	200
Understanding Predefined JavaScript Objects	201
The Navigator Object	201
The History Object	204
Chapter 8 Self Test	206
9 The Document Object	209
Defining the Document Object	210
Using the Document Object Model	210
Using the Properties of the Document Object	211
Collections	211
The cookie Property	214
The dir Property	214
The lastModified Property	214
The referrer Property	215
The title Property	216
The URL Property	217
The URLUnencoded Property	217

Using the Methods of the Document Object	218
The get Methods for Elements	218
The open() and close() Methods	223
The write() and writeln() Methods	225
Using DOM Nodes	225
DOM Node Properties	226
DOM Node Methods	228
Try This 9-1: Add a DOM Node to the Document	233
Creating Dynamic Scripts	234
Styles in JavaScript	234
Simple Event Handling	236
Coding a Dynamic Script	237
Try This 9-2: Try Out Property Changes	238
Chapter 9 Self Test	239
10 Event Handlers	243
What Is an Event Handler?	244
Why Event Handlers Are Useful	244
Understanding Event Handler Locations and Uses	245
Using an Event Handler in an HTML Element	245
Using an Event Handler in the Script Code	246
Learning the Events	248
The Click Event	250
Focus and Blur Events	251
The Load and Unload Events	253
The Reset and Submit Events	255
The Mouse Events	256
The Keyboard Events	257
Try This 10-1: Focus and Blur	258
Other Ways to Register Events	259
The addEventListener() Method	260
The attachEvent() Method	261
The Event Object	261
DOM and Internet Explorer: DOM Level 0 Registration	262
Using event with Modern Event Registration	262
Properties and Methods	263
Event Information	264
Try This 10-2: Using addEventListener()	265
Creating Scripts Using Event Handlers	265
Show Hidden Content	266
Change Content	268
Chapter 10 Self Test	272

11 Window Object	275
Window: The Global Object	276
Using the Properties of the Window Object	277
The closed Property	278
The frames Property	278
The innerWidth and innerHeight Properties	278
The length Property	279
The location Property	279
The name Property	280
The opener Property	280
The parent, self, and top Properties	281
The status and defaultStatus Properties	281
Try This 11-1: Use the location and innerWidth Properties	281
Using the Methods of the Window Object	282
The alert(), prompt(), and confirm() Methods	282
The find() Method	286
The home() Method	286
The print() Method	287
The setInterval() and clearInterval() Methods	287
The setTimeout() and clearTimeout() Methods	289
Try This 11-2: Use the setTimeout() and confirm() Methods	291
The Main Window and New Windows	292
The Tale of Pop-up Windows	292
Opening New Windows	293
Closing New Windows	295
Moving, Resizing, and Scrolling New Windows	297
The resizeBy() and resizeTo() Methods	300
The scrollBy() and scrollTo() Methods	303
Chapter 11 Self Test	303
12 Math, Number, and Date Objects	305
Using the Math Object	306
What Is the Math Object?	306
How the Math Object Is Useful	306
Properties	306
Methods	309
Try This 12-1: Display a Random Link on a Page	322
Understanding the Number Object	323
Properties	323
Methods	325
Using the Date Object	328
Properties and Methods	328
Methods That Get Values	330
Methods That Set Values	333

Other Methods	334
How About Some Date Scripts?	335
Try This 12-2: Create a JavaScript Clock	338
Chapter 12 Self Test	341
13 Handling Strings	343
Introduction to the String Object	344
The String Object	344
The String Literal	344
What's the Difference?	345
Using the Properties and Methods of the String Object	346
The length Property	346
Methods of the String Object	346
Try This 13-1: Use indexOf() to Test an Address	358
Using Cookies	359
Setting a Cookie	360
Reading a Cookie	362
Try This 13-2: Remember a Name	364
Using Regular Expressions	365
Creating Regular Expressions	365
Testing Strings Against Regular Expressions	366
Adding Flags	368
Creating Powerful Patterns	368
Grouping Expressions	372
The replace(), match(), and search() Methods	372
More Information	375
Chapter 13 Self Test	375
14 JavaScript and Forms	379
Accessing Forms	380
Using the forms Array	380
Using Form Names	383
Using an ID	385
Using the Properties and Methods of the Form Object	385
Properties	386
Methods	389
Ensuring the Accessibility of Forms	390
Using Proper Element and Label Order	391
Using <label></label> Tags	391
Using <fieldset></fieldset> Tags	392
Not Assuming Client-Side Scripting	392
Validation	393
Simple Validation	393

Techniques	394
Check Boxes and Radio Buttons	395
Try This 14-1: Request a Number	398
HTML5 and Forms	399
New Elements	399
New Input Types	403
New Attributes	403
HTML5 Form Validation	405
Try This 14-2: Validate a Phone Number with HTML5 or JavaScript	407
Chapter 14 Self Test	408
15 An Introduction to Advanced Techniques	411
Working with Images	412
Rollovers	412
Try This 15-1: A More Accessible Rollover	414
JavaScript and Frames	415
Purpose of Frames	415
Accessing Frames	416
Breaking Out of Frames	419
Debugging Scripts	419
Types of Errors	419
Alerts and Using the Console	422
Using a Lint Tool	423
Browser Developer Tools	425
JavaScript and Accessibility	426
Separate Content from Presentation	426
Enhancing Content	428
Try This 15-2: Make This Code Accessible	429
JavaScript Security	430
Security and Signed Scripts	430
Page Protection	431
AJAX and JSON	433
AJAX	433
JSON	439
Chapter 15 Self Test	442
16 JavaScript Libraries, HTML5, and Harmony	445
Using jQuery	446
Obtaining jQuery	446
Getting Started: document.ready()	447
Using Selectors	447
Altering Classes	449
Methods for Effects	451
Further Reading	453
Try This 16-1: Using jQuery to Create Effects	453

Other JavaScript Libraries	454
jQuery Mobile	454
php.js	454
node.js	454
MooTools	455
Three.js	455
JavaScript and HTML5	456
The <canvas> Element	456
Drag and Drop	462
Try This 16-2: Drag and Drop	466
ECMAScript Harmony	467
The const and let Keywords	467
Default Argument Values	470
Classes	471
More on Harmony	472
Further Reading	472
Need Help?	472
Chapter 16 Self Test	473
A Answers to Self Tests	477
Chapter 1: Introduction to JavaScript	478
Chapter 2: Placing JavaScript in an HTML File	478
Chapter 3: Using Variables	479
Chapter 4: Using Functions	479
Chapter 5: JavaScript Operators	480
Chapter 6: Conditional Statements and Loops	481
Chapter 7: JavaScript Arrays	481
Chapter 8: Objects	482
Chapter 9: The Document Object	483
Chapter 10: Event Handlers	483
Chapter 11: Window Object	484
Chapter 12: Math, Number, and Date Objects	485
Chapter 13: Handling Strings	485
Chapter 14: JavaScript and Forms	486
Chapter 15: An Introduction to Advanced Techniques	487
Chapter 16: JavaScript Libraries, HTML5, and Harmony	487
Index	489

Acknowledgments

I would like to begin by thanking my wonderful wife, Heather Pollock, for all of her love, support, and encouragement in all I do. I love you! I would also like to thank my two daughters, Eva and Elizabeth. I love both of you!

I would like to thank my parents, Bruce and Joy Anderson, for their love and guidance, and for always supporting my endeavors.

I would like to thank Dr. J. D. and Linda Andrews for their love, guidance, and support.

In addition I would like to thank John and Betty Hopkins (grandparents), James D. and Livian Anderson (grandparents), Clifton and Juanita Idom (grandparents), Richard Pollock (brother) and family, Misty Castleman (sister) and family, Warren Anderson (brother) and family, Jon Andrews (brother) and family, Lisa and Julian Owens (aunt/uncle) and family, and every aunt, uncle, cousin, or other relation in my family. All of you have been a great influence in my life.

I would like to thank all of my editors at McGraw-Hill/Professional for their outstanding help and support throughout the writing of this book. Thanks to Brandi Shailer, Ryan Willard, Amanda Russell, and to all of the editors who worked on each edition of the book.

Thanks to my technical editor, Christie Sorenson, for editing and checking over all of the technical aspects of the book, and helping me provide clear explanations of the topics that are covered.

I would like to thank God for the ability He has given me to help and teach people by my writing. “In all your ways acknowledge Him, and He shall direct your paths.” (Proverbs 3:6).

This page intentionally left blank

Introduction

Welcome to *JavaScript: A Beginner's Guide, Fourth Edition*! Years ago, I was surfing the Web and noticed that people were publishing pages about themselves and calling them homepages. After viewing a number of these, I decided to create a homepage myself. I had no idea where to begin but, through trial and error, I figured out how to code HTML and publish my documents on a Web server. Over time, I saw some interesting effects used on other homepages (like alert messages that popped up out of nowhere or images that would magically change when I moved my mouse over them). I was curious and just *had* to know what was being done to create those effects. Were these page creators using HTML tags I did not know about?

Eventually, one site revealed what they were using to create those effects: *JavaScript*. I went in search of information on it, and came across a few tutorials and scripts on the Web. Since I had programmed in other languages (such as a relatively obscure language called Ada), I was able to catch on to JavaScript fairly quickly by looking at these tutorials and scripts.

I learned enough that I decided to create a Web site that would teach HTML and JavaScript to beginners. As soon as I began the project, I received questions from visitors that were way over my head—forcing me to dig deeper and learn more about JavaScript. As a result, I became completely familiar with this scripting language and what it can do. Not only can you add fun effects to a Web page, you can create scripts that will perform useful tasks, like validate form input, add navigational elements to documents, or react to user events.

The goal of this book is to help you to learn the basics of the JavaScript language with as little hair pulling and monitor smashing as possible. You do not need any prior programming experience to learn JavaScript from this book. All you need is knowledge of HTML and/or XHTML, Cascading Style Sheets (CSS), and how to use your favorite text editor and Web browser (see Chapter 1 for more information).

What This Book Covers

The 16 chapters of this book cover specific topics on the JavaScript language. The first two chapters cover the most basic aspects of the language: what it is, what you need to know to begin using JavaScript, and how to place JavaScript into an HTML file. The middle of the book (Chapters 3–14) covers beginning JavaScript topics from variables all the way to using JavaScript with forms. The final two chapters (Chapters 15–16) introduce some advanced techniques, and point you toward resources if you want to learn more about JavaScript once you have completed the book.

This book includes a number of special features in each chapter to assist you in learning JavaScript. These features include

- **Key Skills & Concepts** Each chapter begins with a set of key skills and concepts that you will understand by the end of the chapter.
- **Ask the Expert** The Ask the Expert sections present commonly asked questions about topics covered in the preceding text, with responses from the author.
- **Try This** These sections get you to practice what you have learned using a hands-on approach. Each Try This will have you code a script through step-by-step directions on what you need to do to in order to accomplish the goal. You can find solutions to each project on the McGraw-Hill/Professional Web site at www.mhprofessional.com/computingdownload.
- **Notes, Tips, and Cautions** Notes, Tips, and Cautions call your attention to noteworthy statements that you will find helpful as you move through the chapters.
- **Code** Code listings display example source code used in scripts or programs.
- **Callouts** Callouts display helpful hints and notes about the example code, pointing to the relevant lines in the code.
- **Self Test** Each chapter ends with a Self Test, a series of 15 questions to see if you have mastered the topics covered in the chapter. The answers to each Self Test can be found in the appendix.

That is it! You are now familiar with the organization and special features of this book to start your journey through JavaScript. If you find that you are stuck and need help, feel free to get online and visit the JavaScript discussion forums on the Web Xpertz Web site at www.webxpertz.net/forums. The forums will allow you to interact with other JavaScript coders who may be able to help you with your questions. If you would like to contact me, you can send me a message on my Web site (www.scripttheweb.com/about/contact.html) or you can find me on Twitter (@ScripttheWeb).

Now it is time to learn JavaScript. Get ready, get set, and have fun!



Chapter 1

Introduction to JavaScript

Key Skills & Concepts

- Using Text Editors, WYSIWYG Editors, and Web Browsers
 - Defining JavaScript
 - Differences Between JavaScript and Other Languages
-

Welcome to *JavaScript: A Beginner's Guide, Fourth Edition*! You're obviously interested in learning JavaScript, but perhaps you're not sure what you need to know to use it. This chapter answers some basic questions about what JavaScript is, discusses its advantages and limitations, explains how you can use it to create more dynamic and inviting Web pages, and provides a brief history of the language.

JavaScript is ubiquitous on the World Wide Web. You can use JavaScript both to make your Web pages more interactive, so that they react to a viewer's actions, and to give your Web pages some special effects (visual or otherwise).

JavaScript often gets thrown in with Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS) as one of the recommended languages for beginning Web developers (whether you build Web sites for business or pleasure). Of course, you can build a Web page by using only HTML and CSS, but JavaScript allows you to add additional features that a static page of HTML can't provide without some sort of scripting or programming help.

What You Need to Know

Before you begin learning about JavaScript, you should have (or obtain) a basic knowledge of the following:

- HTML and Cascading Style Sheets (CSS)
- Text editors
- Web browsers
- The different versions of JavaScript

If you have this basic knowledge (the different versions of JavaScript will be discussed in this chapter), then you'll do just fine as you work through this book. Knowing another programming/scripting language or having previous experience with JavaScript isn't required. This book is a *beginner's* guide to JavaScript.

If you think you don't have enough experience in one of the aforementioned areas, a closer look at each one may help you decide what to do.

Basic HTML and CSS Knowledge

While you don't need to be an HTML guru, you do need to know where to place certain elements (like the head and body elements) and how to add your own attributes. This book will reference scripts in the head section (between the <head> and </head> tags) and the body section (between the <body> and </body> tags).

Sometimes, you will also need to add an attribute to a tag for a script to function properly. For example, you may need to name a form element using the id attribute, as shown in the following code:

```
<input type="text" id="thename">
```

If you know the basics of using tags and attributes, the HTML portion shouldn't pose any problems in learning JavaScript.

If you don't have a basic knowledge of HTML, you can learn it fairly quickly through a number of media. For example, you can buy a book or look for some helpful information on the Web. A good book is *HTML: A Beginner's Guide, Fourth Edition* by Wendy Willard (McGraw-Hill Professional, 2009). To find information about HTML on the Web, check out these sites: www.scripttheweb.com/html and www.w3.org/wiki/The_basics_of_HTML.

Occasionally, you will need to use CSS to add or change presentation features on a Web page. We will mainly use CSS for the purposes of dynamically changing CSS properties via JavaScript in this book. Two good places to learn CSS are www.scripttheweb.com/css/ and www.w3.org/wiki/CSS_basics.

Basic Text Editor and Web Browser Knowledge

Before jumping in and coding with JavaScript, you must be able to use a text editor or HTML editor, and a Web browser. You'll use these tools to code your scripts.

Text Editors

A number of text editors and HTML editors support JavaScript. If you know HTML, you've probably already used an HTML editor to create your HTML files, so you might not have to change.

However, some HTML editors have problems related to adding JavaScript code (such as changing where the code is placed or altering the code itself when you save the file). This is more often the case when using WYSIWYG (What You See Is What You Get) editors. It is best to use an HTML editor that allows the option to write your own code (such as Adobe Dreamweaver), a code editor such as NetBeans, or a plain text editor. Some examples of text editors are Notepad, TextPad, and Simple Text.

Web Browsers

Again, if you've been coding in HTML, you probably won't need to change your browser. However, some browsers have trouble with the newer versions of JavaScript. The choice of

Web browser is ultimately up to you, as long as it's compatible with JavaScript. I recommend one of the following browsers to test your JavaScript code:

- Microsoft Internet Explorer version 9.0 or later (or 8 if you are not able to upgrade on your operating system, for example, Windows XP)
- Mozilla Firefox version 14.0 or later
- Google Chrome version 20.0 or later
- Opera version 12.0 or later

New versions of these browsers continue to be produced. The newest versions will continue to support more features.

To give you an idea of what some browsers look like, Figure 1-1 shows a Web page when viewed in Microsoft Internet Explorer, and Figure 1-2 shows the same page when viewed in Mozilla Firefox.

If you have an older browser and you can't upgrade, a number of features (mostly discussed later in the book) may not work in that browser. Even so, the book can still help you learn the JavaScript language itself, so you don't need to give up if you have an older browser.

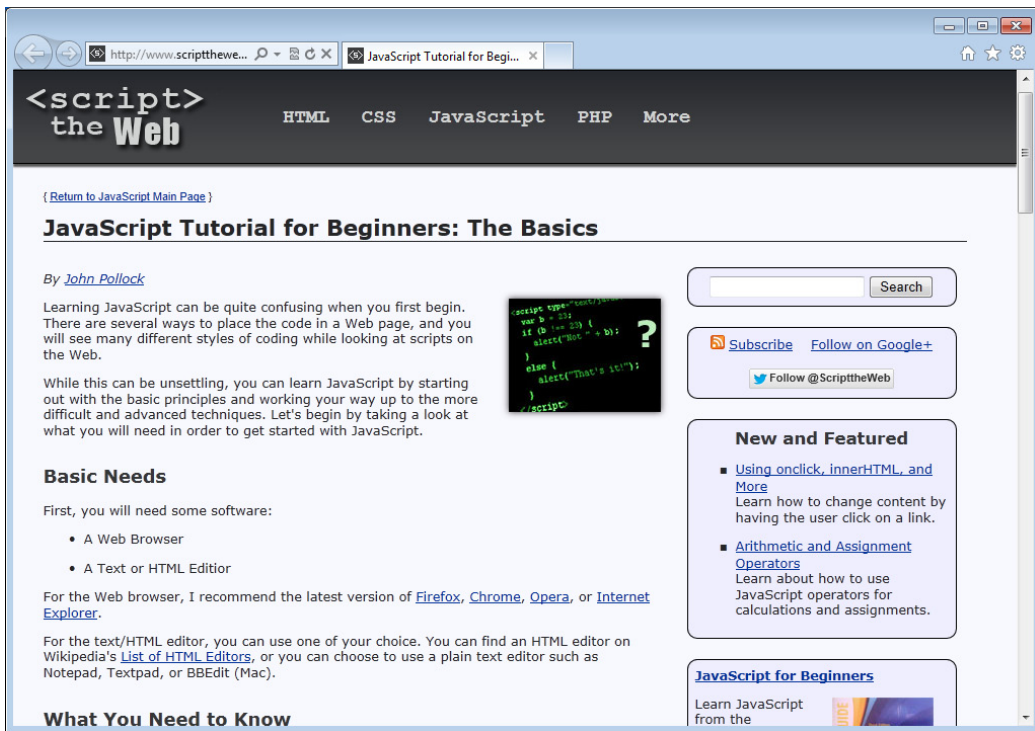


Figure 1-1 A Web page viewed in Microsoft Internet Explorer



Figure 1-2 A Web page viewed in Mozilla Firefox

NOTE

Even if you have one of the latest browsers, your viewers may not, so it is always appropriate to understand what features may not be supported in older browsers. This book will cover how to handle a number of these issues.

Which Version?

The version of JavaScript being used by a browser (or other software) can be a difficult number to identify, since different browsers may have different names for the language (for example, JScript in Microsoft Internet Explorer) and each of these can follow its own set of version numbers. To find out what standard conventions are being followed, it is best to look at what version of ECMAScript is supported in each browser. You can see what versions are supported by each browser at en.wikipedia.org/wiki/ECMAScript#Dialects.

ECMAScript is the international standard name and specification used for the JavaScript language, so it's not a new language but a standard that is set for JavaScript, JScript, and other implementations of the language. For more on ECMAScript, see www.ecmascript.org/docs.php.

At the time of this writing, the browsers recommended earlier in this chapter should support at least ECMAScript 5.

Remember, It's Not Java

JavaScript and Java are two different languages. Java is a programming language that must be compiled (running a program through software that converts the higher-level code to machine language) before a program can be executed. More information on the Java language can be found at docs.oracle.com/javase/tutorial/.

Similarities to Other Languages

JavaScript does have similarities to other programming and scripting languages. If you have experience with Java, C++, or C, you'll notice some similarities in the syntax, which may help you to learn more quickly. Because it's a scripting language, JavaScript also has similarities to languages like Perl—which can also be run through an interpreter rather than being compiled.

If you have programming or scripting experience in any language, it will make learning JavaScript easier—but it isn't required.

Ask the Expert

Q: You mentioned that I could use a text editor or HTML editor of my choice, but I'm not quite sure what that means. What is a text editor and where can I find one?

A: A text editor is a program that you can use to save and edit written text. Text editors range from simple to complex, and a number of choices are available: Notepad, WordPad, and Simple Text, to name a few. You can also purchase and download some from the Web, like NoteTab or TextPad.

An HTML editor is either a more complex text editor or an editor that allows you to add code by clicking buttons or by other means—often called a What You See Is What You Get (WYSIWYG) editor. I recommend a plain text editor or an HTML editor that doesn't change any code you add to it manually. Some examples of HTML editors are Adobe Dreamweaver and Softpress Freeway.

For the purposes of JavaScript coding, you may decide to use a more code-oriented program that can offer features such as code highlighting, completion, debugging tools, and more, such as NetBeans IDE.

Q: What exactly do I need to know about using a text editor?

A: Basically, you only need to know how to type plain text into the editor, save the file with an .html, .css, or .js extension, and be able to open it again and edit it if necessary. Special features aren't needed because HTML, CSS, and JavaScript files are made up of plain text.

Q: What do I need to know about using a browser?

A: All you absolutely need to know is how to open a local HTML file on your computer (or on the Web) and how to reload a page. If you don't know how to open an HTML file from your own computer, open your browser and go to the File menu. Look for an option that says something like Open, or Open File, and select it. You should be able to browse for the file you want to open, as you would with other programs. The following illustration shows where the option is in Microsoft Internet Explorer:

**Q: Where do I get those browsers you mentioned?**

A: Here are links for the browsers:

- **Microsoft Internet Explorer** www.microsoft.com/ie
- **Mozilla Firefox** www.mozilla.com/firefox
- **Google Chrome** www.google.com/chrome/
- **Opera** www.opera.com

Beginning with JavaScript

JavaScript came about as a joint effort between Netscape Communications Corporation and Sun Microsystems, Inc. The news release of the new language was issued on December 4, 1995, back when Netscape Navigator 2.0 was still in its beta version. JavaScript version 1.0 became available with the new browser. (Before its release as JavaScript, it was called LiveScript.)

JavaScript is a prototype-based, client-side scripting language that can be used in numerous environments. To make sense of such a definition, let's look at its important parts one by one.

Prototype-Based

Prototype-based means that JavaScript is an object-oriented programming language that can use items called *objects*. However, the objects are not *class-based*, so no distinction is made between a class and an instance; instead, objects inherit from other objects via the prototype property. You'll learn how to work with JavaScript objects in Chapter 8. You don't need to understand them in detail until you know a few other features of the language.

Client-Side

Client-side means that JavaScript runs in the *client* (software) that the viewer is using, rather than on the Web server of the site serving the page. In this case, the client will most often be a Web browser (though ECMAScript can be run in other environments as well, such as Adobe Acrobat, Adobe Flash, and others). To make more sense of this, let's take a look at how a server-side language works and how a client-side language works.

Server-Side Languages

A *server-side language* needs to get information from the Web page or the Web browser, send it to a program that is run on the host's server, and then send the information back to the browser. Therefore, an intermediate step must send and retrieve information from the server before the results of the program are seen in the browser.

A server-side language often gives the programmer options that a client-side language doesn't have, such as saving information on the Web server for later use, or using the new information to update a Web page and save the updates.

However, a server-side language is likely to be limited in its ability to deal with special features of the client that can be accessed with a client-side language (like the width of the browser window or the contents of a form before it's submitted to the server).

Client-Side Languages

A *client-side language* is run directly through the client being used by the viewer. In the case of JavaScript, the client is typically a Web browser. Therefore, JavaScript is run directly in the Web browser and doesn't need to go through the extra step of sending and retrieving information from the Web server.

With a client-side language, the browser reads and interprets the code, and the results can be given to the viewer without getting information from the server first. This process can make certain tasks run more quickly.

A client-side language can also access special features of a browser window that may not be accessible with a server-side language. However, a client-side language lacks the ability to save files or updates to files on a Web server as a server-side language can.

NOTE

Using the XMLHttpRequest object allows JavaScript to send and request data from the server. This will be covered briefly in Chapter 15.

A client-side language is useful for tasks that deal with the content of a document or that allow information to be validated before it is sent to a server-side program or script. For instance, JavaScript can change the content of one or more elements on a Web page when the user clicks a link or presses a button (many other user actions can also be activated).

JavaScript can also be used to check the information entered into a form before the form is sent to a server-side program to be processed. This information check can prevent strain on the Web server by preventing submissions with inaccurate or incomplete information. Rather than running the program on the server until the information is correct, that data can be sent to the server just once with correct information.

NOTE

While JavaScript is able to *help* validate information sent to the server, it cannot replace server-side validation since users may have JavaScript disabled or unavailable in the device being used (which allows them to bypass the JavaScript validation). For security reasons, you should always use server-side validation, regardless of whether or not you incorporate JavaScript validation.

Scripting Language

A scripting language doesn't require a program to be compiled before it is run. All the interpretation is done on-the-fly by the client.

With a regular programming language, before you can run a program you have written, you must compile it using a special compiler to be sure there are no syntax errors. With a scripting language, the code is interpreted as it is loaded in the client. Thus, you can test the results of your code more quickly. However, errors won't be caught before the script is run and could cause problems with the client if it can't handle the errors well. In the case of JavaScript, the error handling is up to the browser or other client being used by the viewer.

Putting It All Together

With all this in mind, you might wonder how JavaScript is run in a browser. You might wonder where to write your JavaScript code and what tells the browser it is different from anything else on a Web page. The answers are general for now, but the next chapter provides more details.

JavaScript runs in the browser by being added into an existing HTML document (either directly or by referring to an external script file). You can add special tags and commands to the HTML code that will tell the browser that it needs to run a script. When the browser sees

these special tags, it interprets the JavaScript commands and will do what you have directed it to do with your code. Thus, by simply editing an HTML document, you can begin using JavaScript on your Web pages and see the results.

For example, the following code adds some JavaScript to an HTML file that writes some text onto the Web page. Notice the addition of `<script>` and `</script>` tags. The code within them is JavaScript.

```

<html>
<body>
<script>
document.write("This writes text to the page");
</script>
</body>
</html>

```

This tag tells the browser that JavaScript follows

This line writes the text inside the quote marks on the page

This line tells the browser that this is the end of the script

The next chapter looks at how to add JavaScript in an HTML file by using the `<script>` and `</script>` HTML tags. This will be your first step on the road to becoming a JavaScript coder!

Online Resources

To find additional information online to help you with JavaScript, here are some useful resources:

- A place to find tutorials with working examples of the results: www.scripttheweb.com/js/
- An excellent tutorial site that includes cut-and-paste scripts: www.javascriptkit.com
- A place where you can address questions about JavaScript to fellow coders: www.webxpertz.net/forums

Try This 1-1 Use JavaScript to Write Text

pr1_1.html

This project shows you JavaScript in action by loading an HTML document in your browser. The script writes a line of text in the browser using JavaScript.

Step by Step

1. Copy and paste the code shown here into your text editor:

```

<html>
<body>
<script>
document.write("This text was written with JavaScript!");
</script>
</body>
</html>

```

2. Save the file as `pr1_1.html` and open it in your Web browser. You should see a single line of text that was written with JavaScript. (To open a file in your Web browser, go to the File menu and look for an option that says something like Open, or Open File, and select it. You should be able to browse for the file you want to open as you would with other programs.)

Try This Summary

In this project, you copied and pasted a section of code into a text editor and saved the file. When you opened the saved file in your Web browser, a line of text was displayed in the browser. This text was written in the browser window using JavaScript. You will see more about how this type of script works in Chapter 2.



Chapter 1 Self Test

1. You must know which of the following to be able to use JavaScript?
 - A. Perl
 - B. C++
 - C. HTML
 - D. SGML
2. Which of the following is something you should have to use JavaScript?
 - A. A Web browser
 - B. A C++ compiler
 - C. A 500GB hard drive
 - D. A DVD-RW drive
3. The choice of a Web browser is up to you, as long it's compatible with _____.
 - A. Flash
 - B. VBScript
 - C. JavaScript
 - D. Windows XP
4. JavaScript and Java are the same language.
 - A. True
 - B. False

5. _____ is the international standard name and specification used for the JavaScript language.
 - A. JScript
 - B. LiveScript
 - C. ECMAScript
 - D. ActionScript
6. JavaScript has similarities to other programming and scripting languages.
 - A. True
 - B. False
7. Before its release as JavaScript, JavaScript was called _____.
 - A. Java
 - B. JavaCup
 - C. LiveScript
 - D. EasyScript
8. JavaScript is _____.
 - A. prototype-based
 - B. class-based
 - C. object deficient
 - D. not a language that uses objects
9. A client-side language is run directly through the _____ being used by the viewer.
 - A. server
 - B. client
 - C. monitor
 - D. lawyer
10. How can a client-side language help when using forms on a Web page?
 - A. It can save the information on the server.
 - B. It can validate the information before it is sent to the server.
 - C. It can update a file and save the file with the new information.
 - D. It can't help at all.

11. A _____ language doesn't require a program to be compiled before it is run.
 - A. programming
 - B. server-side
 - C. scripting
 - D. computer
12. With a scripting language, the code is interpreted as it is loaded in the client.
 - A. True
 - B. False
13. In JavaScript, what handles errors in a script?
 - A. The Web server
 - B. A compiler
 - C. A program on the Web server
 - D. The client/Web browser
14. How is JavaScript added to a Web page?
 - A. It isn't. It must be compiled and loaded separately.
 - B. It is taken from a compiled program on the server.
 - C. You place the code in a file by itself and open that file.
 - D. It is added to an HTML document.
15. What is added to a Web page to insert JavaScript code?
 - A. `<script>` and `</script>` HTML tags
 - B. The JavaScript code word
 - C. `<javascript>` and `</javascript>` HTML tags
 - D. `<java>` and `</java>` HTML tags

This page intentionally left blank



Chapter 2

Placing JavaScript in an HTML File

Key Skills & Concepts

- Using the HTML Script Tags
- Creating Your First Script
- Using External JavaScript Files
- Using JavaScript Comments

Now that you have been introduced to JavaScript, you're ready to start coding. Since JavaScript code is run from HTML documents, you need to know how to tell browsers to run your scripts. The most common way to set off a script is to use the HTML `<script>` and `</script>` tags in your document. You can place your script tags in either the head or the body section of an HTML document.

This chapter first shows you how to use the script tags to begin and end a segment of JavaScript code. Then, you will get started creating and running your first scripts. At the end of the chapter, you will learn how to add JavaScript comments to document your scripts.

Using the HTML Script Tags

Script tags are used to tell the browser where code for a scripting language will begin and end in an HTML document. In their most basic form, script tags appear just like any other set of HTML tags:

```
<script> ← Tells the browser where  
JavaScript code here script code begins  
</script> ← Tells the browser where  
script code ends
```

As you can see, there is the opening `<script>` tag, the JavaScript code, and then the closing `</script>` tag. When you use just the basic opening and closing tags like this, almost all browsers will assume that the scripting language to follow will be JavaScript.

In HTML, the script tag is not case sensitive. However, in XHTML, the script tag must be in lowercase. JavaScript is case sensitive in all versions, so you will need to be more careful with it. In this book, I will use HTML5 for the HTML code (even though HTML5 is not case sensitive, I will write the tag and attribute names in lowercase). For the JavaScript code, I will use the case that is needed for it to function correctly.

The `<script>` tag has six possible attributes: *type*, *language* (deprecated), *charset*, *src*, *defer*, and *async*. These attributes give the browser additional information about when the script should load, the scripting language, and the location of an external JavaScript file (if any).

Identifying the Scripting Language

The scripting language between the opening and closing script tags could be JavaScript, VBScript, or some other language, though JavaScript is almost always set as the default scripting language in browsers. If desired, you can explicitly identify JavaScript as the scripting language by adding the type attribute with the value of “text/javascript” to the opening script tag:

```
<script type="text/javascript"> ← Tells the browser the scripting  
JavaScript code here           language will be JavaScript  
</script>
```

NOTE

The type attribute in the opening script tag is required in XHTML in order for the Web page to validate, but is optional in HTML.

In the past, the language attribute was used to identify the scripting language, but is ignored in modern browsers and will cause the page to be invalid in XHTML and HTML5. It should no longer be used.

The charset attribute, which allows for the character set of the JavaScript code to be specified, is not recognized by most browsers and is not recommended.

Calling External Scripts

Script tags allow you to call an external JavaScript file in your document. An *external JavaScript file* is a text file that contains nothing but JavaScript code, and it is saved with the .js file extension. By calling an external file, you can save the time of coding or copying a long script into each page in which the script is needed. Instead, you can use a single line on each page that points to the JavaScript file with all of the code.

You can call external scripts by adding the src (source) attribute to the opening script tag:

```
<script type="text/javascript" src="yourfile.js"></script>
```

This example calls a JavaScript file named yourfile.js from any page on which you place this tag. Be sure there are no spaces or code between the opening and closing script tags, as this may cause the script call to fail.

If the script is extremely long, using the src attribute to add the script to multiple pages can be much quicker than inserting the entire code on each page. Also, the browser will cache the external JavaScript file the first time it is loaded, making subsequent Web pages that use the script render faster. Using an external script is also helpful when dealing with page validation and when trying to keep script code separated from markup (HTML) code.

By default, script files are loaded in the order in which they are placed in the HTML code (synchronously). There are some options for altering this behavior, which are described in the next section.

Specifying When the Script Should Load

The last two attributes, `defer` and `async`, allow you to specify when an external script should be loaded. These attributes are not fully supported by older browsers, or may behave differently, so be aware that an older browser may not execute the script when it is expected to do so.

The `defer` Attribute

The `defer` attribute allows you to specify that an external script file should be loaded, but should not execute until the page has completed parsing (the `</html>` tag has loaded). The following `<script>` tag would defer the execution of the external JavaScript code:

```
<script type="text/javascript" src="file.js" defer></script>
```

NOTE

If you are using XHTML, set this attribute using `defer="defer"`.

Support for this attribute is available in Internet Explorer 4+, Firefox 3.5+, and Chrome 7+. Internet Explorer 4–7 will allow this attribute to work on inline scripts as well, but versions 8 and above only support this attribute on external scripts as other browsers do.

The `async` Attribute

When the `async` attribute is set, the page can continue to load without waiting for the script to load, and the script will execute before the document completes loading. Here is an example:

```
<script type="text/javascript" src="file.js" async></script>
```

NOTE

If you are using XHTML, set this attribute using `async="async"`.

Support for this attribute is available in Firefox 3.5+ and Chrome 7+.

Using `<noscript></noscript>` Tags

One way of providing alternate content for those viewers without JavaScript (or with JavaScript turned off) is to use the `<noscript></noscript>` tags. The `<noscript></noscript>` tags may be placed anywhere in the HTML document and can contain any content needed for those viewers browsing without JavaScript. For example:

```
<script type="text/javascript">
  document.write("The color is red.");
</script>
<noscript>
  The color is red.
</noscript>
```

Displays for those viewers
with JavaScript

←

Begins noscript content for those
viewers without JavaScript

←

Ends noscript content

This example displays the phrase “The color is red.” to the viewer either through JavaScript or through the text within the `<noscript></noscript>` tags.

CAUTION

Some older browsers may not handle the `noscript` tag correctly and won't display the content in either section. If your users have older browsers, another alternative is to display the content on the page and then use JavaScript to enhance the content for those who are able to display it with JavaScript on.

The `<noscript>` tag can be useful at times, but there are often better ways to provide the same content to those without JavaScript (avoiding the `document.write()` method, for instance). You will learn more about accessible JavaScript as you progress through this book.

Ask the Expert

Q: Do I always need to use script tags to add JavaScript to a page?

A: It's possible to use *event handlers* that allow you to write short bits of script within the event-handling attribute of an HTML tag. You'll learn about event handlers in Chapter 8.

Q: What about the language attribute?

A: The language attribute once was used to specify the scripting language to be used, and for a time some browsers allowed a JavaScript version to be specified (for example, `language="JavaScript1.2"`). This is no longer the case, and the attribute has been deprecated. Since it is completely ignored in modern browsers and causes pages not to validate in XHTML and HTML5, it should no longer be used.

Q: My page won't validate in XHTML strict (or transitional) when I add a script to it. How do I get the page to validate?

A: If the script contains characters used in XHTML such as `<` (which is used for "less than" in JavaScript but is seen as the beginning of a new tag in XHTML), then the page won't validate with the script directly in the document without adding a CDATA section:

```
<script type="text/javascript">
<![CDATA[ ←————— Begins the CDATA section
var x = 5;
var y = 10;
if (x < y) {
window.alert("x is less than y");
}
}]> ←————— Ends the CDATA section
</script>
```

(continued)

This will allow the page to validate, but because the `<![CDATA[` and `]]>` characters are in the script, the script will no longer work. To fix this, you need JavaScript comments (`/*` and `*/`) around those characters when they are within the script tags:

```
<script type="text/javascript">
/**/ ← Opening and closing JavaScript comments
var x = 5;      are placed around <![CDATA[
var y = 10;
if (x &lt; y) {
window.alert("x is less than y");
}
/*]]&gt;*/ ← Opening and closing JavaScript
&lt;/script&gt;      comments are placed around ]]&gt;</pre>
</div>
<div data-bbox="169 345 846 398" data-label="Text">
<p>As you can see, this can get quite tedious very quickly! Typically, the better option is to use an external script file, which eliminates this problem because only the script tags themselves are needed in the XHTML document.</p>
</div>
<div data-bbox="118 437 482 470" data-label="Section-Header">
<h2>Creating Your First Script</h2>
</div>
<div data-bbox="153 470 873 561" data-label="Text">
<p>Now that you know how to use the HTML script tags to tell browsers about the JavaScript in a document, you're ready to learn how to add the actual JavaScript code between those script tags. The first coding example often given to teach any language is one that writes some sort of text to the default output area, commonly known as a basic "Hello World" script. Following that convention, your first script will write a string of text to a Web page.</p>
</div>
<div data-bbox="153 572 538 600" data-label="Section-Header">
<h3>Writing a "Hello World" Script</h3>
</div>
<div data-bbox="153 599 860 653" data-label="Text">
<p>Rather than write "Hello World," you'll use another line of text for this script: "Yes! I am now a JavaScript coder!" This requires only a single line of code, using the <code>document.write()</code> method, which writes a string of text to the document:</p>
</div>
<div data-bbox="153 663 706 711" data-label="Text">
<pre>&lt;script type="text/javascript"&gt;
  document.write("Yes! I am now a JavaScript coder!");
&lt;/script&gt;</pre>
</div>
<div data-bbox="153 721 879 775" data-label="Text">
<p>Notice the parentheses and the quotation marks around the text. The parentheses are required because the <code>document.write()</code> method is a JavaScript <i>function</i>, which takes an <i>argument</i> contained in parentheses. You will learn more about JavaScript functions in Chapter 4.</p>
</div>
<div data-bbox="153 775 876 830" data-label="Text">
<p>The quotation marks denote a <i>string</i> of text. A string is a type of variable that is defined in JavaScript by placing it inside quotation marks. Chapter 3 provides details on strings and other types of JavaScript variables.</p>
</div>
<div data-bbox="153 830 878 939" data-label="Text">
<p>The last thing to notice about your script is the semicolon at the end of the line. The semicolon signals the end of a JavaScript statement. A <i>statement</i> is a portion of code that does not need anything added to it to be complete in its syntax (its form and order). A statement can be used to perform a single task, to perform multiple tasks, or to make calls to other parts of the script that perform several statements. Most JavaScript statements end with a semicolon, so it is a good idea to get in the habit of remembering to add one.</p>
</div>
```

NOTE

In later chapters, you will see various lines that do not end in semicolons because they open or close a block of code. Also, many scripts you encounter may not end statements with semicolons. JavaScript is lenient about the use of a semicolon in most cases; however, it is best to use the semicolon to end a statement because it can prevent possible errors and aid in debugging (removing errors from) the script later.

So, to write a text string to the page, you use the `document.write()` method, followed by the text, surrounded by quotation marks and enclosed in parentheses. End the line (the statement) with a semicolon. JavaScript will handle the rest of the job.

Creating an HTML Document for the Script

In order to make this example complete and test the script, you need to insert it into an HTML document. First, create the following HTML document with the basic tags (using any text editor you prefer):

```
<!DOCTYPE html>
<html>
<head>
<title>Untitled Document</title>
</head>
<body>
</body>
</html>
```

Save the document as `test1.html` in your text editor. You will call it later with a Web browser to see the results of the script. Next, you'll add the script to this HTML document, so leave the file open.

Inserting the Script into the HTML Document

Now you need to insert the script in the document. Where should it go? You can place a script between the `<head>` and `</head>` tags, or between the `<body>` and `</body>` tags. Since this example writes a text string directly to the page, you want to insert the script between the `<body>` and `</body>` tags, wherever you want the text string to appear. It can come before, after, or between any HTML code on the page.

To make it clear how the script results appear, you'll add HTML code to write lines of text before and after the script. The script tags and the script itself are inserted between those lines. Add the lines shown next between the `<body>` and `</body>` tags:

```
<p>This is the first line, before the script results.</p>
<p>
<script type="text/javascript">
  document.write("Yes! I am now a JavaScript coder!");
</script>
</p>
<p>This line comes after the script.</p>
```

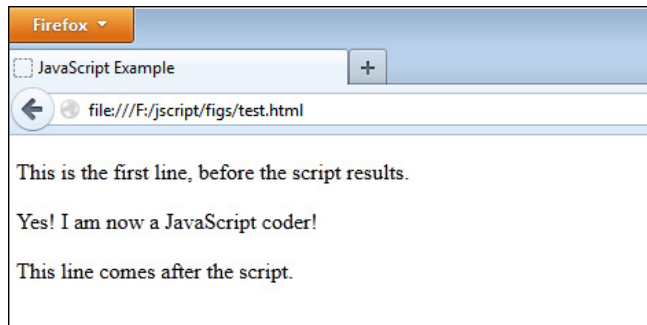


Figure 2-1 The test.html file in a Web browser

Save the test1.html document again. You should now be able to open the document in your Web browser to see the results of the script. Figure 2-1 shows how the text should look in your browser when you load the Web page.

Congratulations, you have now finished your first script!

NOTE

The example code in this section uses the entire HTML document and all of its tags. In order to keep things as relevant as possible, from this point on the example code will use only the HTML tags involved with the scripts rather than the entirety of its tags. Project code may use entire HTML documents as needed.

Ask the Expert

Q: Why is there a dot (.) in the `document.write()` command?

A: Document is one of JavaScript's predefined objects, and `write()` is a predefined method of the document object. The dot puts the object and the method together to make the function work. Chapter 9 explains JavaScript objects, and Chapter 10 is devoted to the document object.

Q: How do I know when to add the script inside the head section and when to add it inside the body section?

A: In the past, JavaScript code was almost always placed inside the head section, which kept it in a separate area from the rest of the HTML code. Modern coding practice is typically to place all JavaScript code in an external .js file and to place the `<script>` tag(s) right before the closing `</body>` tag. This ensures that the HTML page has loaded in the browser (since large scripts can delay the loading of the page if placed elsewhere), giving the user a better experience.

Try This 2-1 Insert a Script into an HTML Document

pr2_1.html

This project gives you practice adding a script to your page. You will create an HTML document and insert a script that displays a short sentence in the browser window when the page loads.

Step by Step

1. Set up an HTML document so that you have a simple file with nothing between the `<body>` and `</body>` tags yet.
2. Put the following line of text into the Web page within a paragraph:

I am part of the HTML document!

3. Insert a `<p>` tag after this line.
4. After the `<p>` tag, insert a script that will write the following line on the page:

This came from my script, and is now on the page!

5. After the script, add a `</p>` tag. Add another opening `<p>` tag.
6. Put the following line of text into the Web page after the last `<p>` tag, and make it emphasized (using `` tags):

I am also part of the HTML document, after the script results!

7. Add a `</p>` tag to complete the paragraph.
8. Here is what your HTML document should look like:

```
<!DOCTYPE html>
<html>
<head>
<title>JavaScript Project 2-1</title>
</head>
<body>
<p>I am part of the HTML document!</p>
<p>
<script type="text/javascript">
  document.write("This came from my script, and is now on the page!");
</script>
</p>
<p><em>I am also part of the HTML document, after the script results!</em></p>
</body>
</html>
```

9. Save the file as `pr2_1.html` and view the page in your browser to see the results.

(continued)

Try This Summary

In this project, you created an HTML file. Using the knowledge that you acquired thus far in this chapter, you inserted within the HTML file a script that writes a specific line of text on the page. When the HTML page is opened in a Web browser, the result of the script is displayed between two lines of text.

Using External JavaScript Files

Now suppose that you want to use your “Hello World” script (the one you created earlier in this chapter) on more than one page, but you do not want to write it out on each page. You can do this by putting the script in an external script file and calling it with the `src` attribute of the script tag. For this method, you need to create a JavaScript text file to hold your script. You also need one or more HTML files into which you will place the script tags to call your external script file.

Creating a JavaScript File

For this example, you will create a JavaScript file that contains only one line. For practical applications, you would use this approach for lengthier scripts—the longer the script is, the more useful this technique becomes (especially if you are trying to validate your Web pages or you are separating your script code from your markup).

Open a new file in your text editor and insert only the JavaScript code (the `document.write()` statement) itself. The script tags are not needed in the external JavaScript file. The file should appear like this:

```
document.write("Yes! I am now a JavaScript coder!");
```

Save the file as `jsfile1.js` in your text editor. To do this, you may need to use the **Save As** option on the **File** menu and place quotation marks around your filename, as shown in Figure 2-2 (using Notepad with Windows).

Once the file has been saved, you can move on to the next step, which is to create the HTML files in which to use the script.

Creating the HTML Files

You will create two files in which to place your script. The technique should work for any number of HTML files, though, as long as you add the required script tags to each file.

For the first file, create your base HTML document and insert the script tags into the `body` section of the document, using the `src` attribute to point to the `jsfile1.js` file, and add some HTML text to the body of the page to identify it as the first HTML document:

```
<body>
<script type="text/javascript" src="jsfile1.js"></script>
<p>
  This is page 1, and the script works here!
</p>
</body>
```

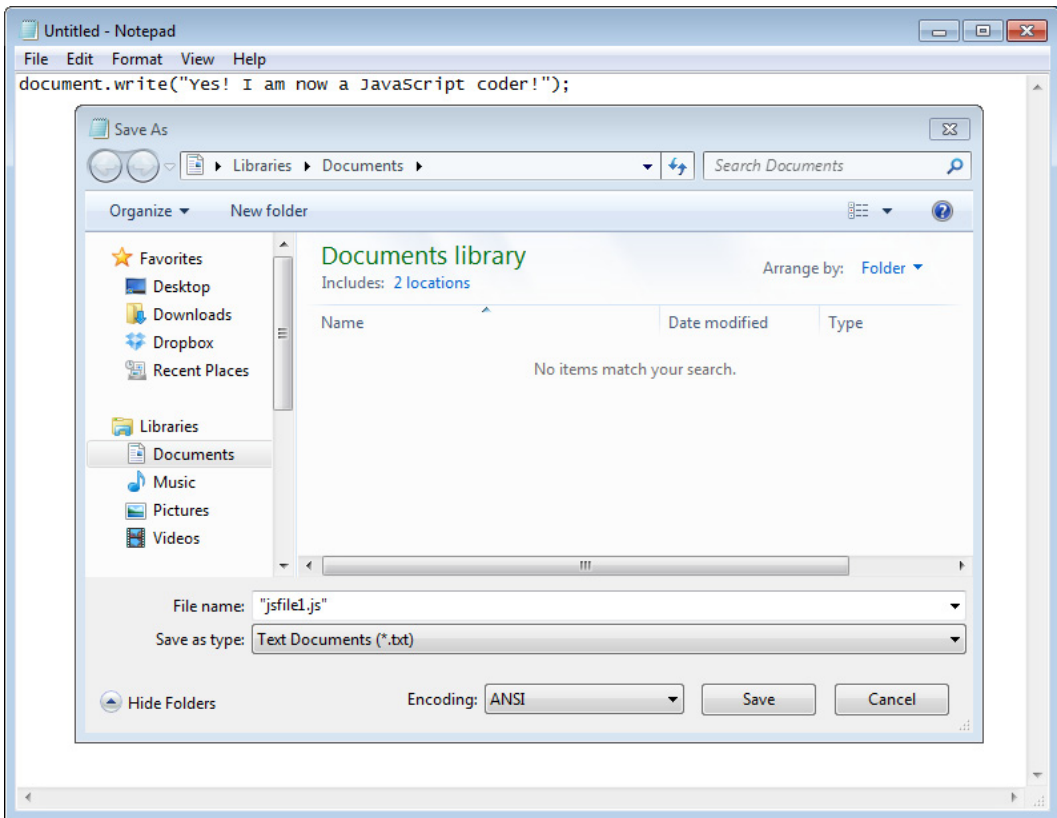


Figure 2-2 An example of saving a file with a .js extension using quote marks so it will save with the correct file extension

Save this file as `jsxt1.html` in your text editor. Be sure to save it in the same directory as your `jsfile1.js` file.

The second HTML document looks the same as the first one, except that the HTML text says that it's page 2:

```
<body>
<script type="text/javascript" src="jsfile1.js"></script>
<p>
  This is page 2, and the script also works here!
</p>
</body>
```

Save this file as `jsxt2.html` in your text editor. Again, be sure to place it in the same directory as the other files.

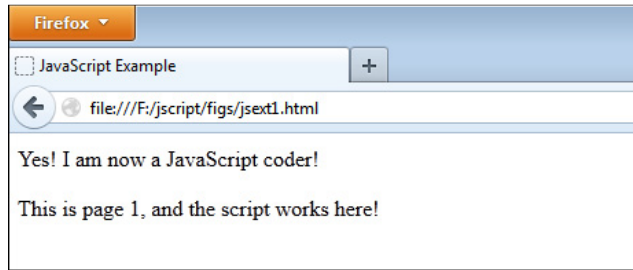


Figure 2-3 The result of calling the script in the `jsxt1.html` file, the first HTML page

Viewing the Pages in Your Browser

Open the `jsxt1.html` file in your Web browser. It should appear as shown in Figure 2-3, with the JavaScript inserted in the page from the external script file.

Next, open the `jsxt2.html` file in your Web browser. It should appear as shown in Figure 2-4, with only the small difference of the text you added to the HTML file to say that this is page 2. The JavaScript should write the same text to this page as it did to the first HTML page.

Although we used a short script in this example, it should give you an idea of how using an external file could be a great time-saver when you have a large script.

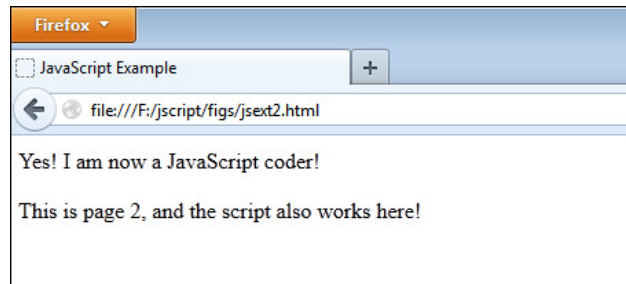


Figure 2-4 The result of calling the script in the `jsxt2.html` file, the second HTML page

Try This 2-2**Call an External Script from an HTML****Document**

```
pr2_2.html  
prjs2_2.js
```

This project will allow you to practice creating external JavaScript files and using them to insert a script into a Web page.

Step by Step

1. Set up a simple HTML document with nothing between the `<body>` and `</body>` tags.
2. Place the following line of text between the body tags of the page:

```
This text is from the HTML document!
```

3. Place a `<p>` tag after this text.
4. Save it as `pr2_2.html`.
5. Create an external JavaScript file that will write the following line when it is executed:

```
I love writing JavaScript and using external files!
```

6. Here is how your JavaScript file should look:

```
document.write("I love writing JavaScript, and using external files!");
```

7. Save the JavaScript file as `prjs2_2.js`.
8. Go back to the HTML document. Place the script tags after the `<p>` tag in the document so that the external JavaScript file will write its sentence on the page.
9. Insert a `</p>` tag after the script tags.
10. The body of your HTML document should look like this:

```
<body>  
This text is from the HTML document!  
<p>  
<script type="text/javascript" src="prjs2_2.js"></script>  
</p>  
</body>
```

11. Save the HTML file and view the results in your browser.

Try This Summary

In this project, you created an HTML page. Using your knowledge of external JavaScript files from the previous section, you created an external JavaScript file and placed the necessary code into the HTML file to include the external JavaScript file. When the HTML file is displayed in a Web browser, a line of plain text is shown, followed by the results of the external JavaScript file.

Using JavaScript Comments

You may need to make notes in your JavaScript code, such as to describe what a line of code is supposed to do. It's also possible that you will want to disable a line of the script for some reason. For instance, if you are looking for an error in a script, you may want to disable a line in the script to see if it is the line causing the error. You can accomplish these tasks by using JavaScript comments. You can insert comments that appear on one line or run for numerous lines.

Inserting Comments on One Line

If you want to add commentary on a single line in your code, place a pair of forward slashes before the text of the comment:

```
// Your comment here
```

In this format, anything preceding the two slashes on that line is “live” code—code that will be executed—and anything after the slashes on that line is ignored. For example, suppose that you wrote this line in your code:

```
document.write("This is cool!"); // writes out my opinion
```

The `document.write()` method will be run by the browser, so the text “This is cool!” will be written to the page. However, the comment after the slashes will be ignored by the browser.

If you place the forward slashes at the beginning of a line, the browser will ignore the entire line. Suppose that you move the slashes in the previous example to be the first items on the line:

```
// document.write("This is cool!"); writes out my opinion
```

In this format, the entire line is ignored, since it begins with the two slashes that represent a JavaScript comment. The text will not be written to the page, since the code will not be executed by the browser. In effect, you are disabling the `document.write()` statement. You may wish to do this if the script containing this line has an error and you want to know whether or not this line is causing the problem.

Adding Multiple-Line Comments

Comments denoted by a pair of forward slashes apply only to the line on which they appear; their effects are cut off at the end of the line. You can span multiple lines with this type of comment by adding the slashes to each line of code, as in this example:

```
// Deleted my commentary.  
// document.write("This is cool!"); writes out my opinion  
// End of commentary deletion.
```

To add comments that span any number of lines without placing slashes on every line, you can use a different comment format: a forward slash followed by an asterisk at the beginning of the

comment, then the text of the comment, and then an asterisk followed by a forward slash at the end of the comment. Here's an example:

```
/*
My script will write some text into my HTML document!
All of this text is ignored by the browser.
*/
document.write("You can see me!");
```

Using this format, you can begin the comment on one line and end it on another line.

Multiple-line comments can be handy when you want to insert lengthier descriptions or other text, but you need to be careful when you use them. Look at this example to see if you can find a problem with it:

```
<script type="text/javascript">
/*
This code won't work for some reason.
document.write("I want someone to see me!");
</script>
```

Did you notice that the closing JavaScript comment symbols are missing? When you use multiple-line comments, you need to be careful to close them. Otherwise, you might accidentally comment out code you need executed! In this example, the comment just keeps going on with no end in sight. To fix this, you need to close the JavaScript comments before the `document.write()` method is used:

```
<script type="text/javascript">
/*
The JavaScript code is now working! This text is hidden.
*/
document.write("Now everyone can see me!");
</script>
```

Note, however, that multiple-line comments cannot be placed inside other multiple-line comments, since the closing comment code for the inner comment will close the comment early. For example, consider this code:

```
/* ← Outside comment started
Comment out some code
document.write("I cannot be seen!"); ← This code won't be executed
/* ← Inside comment started (isn't recognized as a new comment,
since it is already commented out)
document.write("I also cannot be seen."); ← This code also won't be executed
*/ ← This ends the entire comment!
document.write("Oops! I can be seen!"); ← This code will execute!
*/
```

As you can see, the first instance of the ending `*/` causes the entire comment to end, and the code afterward could be executed.

In the preceding examples, you saw how comments can be used to provide some documentation of what to expect from each script. In Chapter 16, you will learn how using comments can help you debug your JavaScript code. For now, you should get in the habit of adding comments to your scripts as short documentation or instructions.



Chapter 2 Self Test

1. What is the purpose of the `<script>` and `</script>` tags?
 - A. To tell the browser where a script begins and ends
 - B. To let the browser know when the script should be loaded
 - C. To point to an external JavaScript file
 - D. All of the above
2. Why would you use the `type` attribute in the opening script tag?
 - A. To let the browser know what type of coder you are
 - B. To ensure the Web page validates when using XHTML
 - C. To create a typing script
 - D. To make sure the script does not make a grammatical error
3. Is JavaScript code case sensitive?
 - A. Yes
 - B. No
4. The `noscript` tag provides _____ for those without _____.
5. An external JavaScript file commonly uses a filename extension of _____.
 - A. `.js`
 - B. `.html`
 - C. `.jav`
 - D. `.java`
6. Which of the following correctly points to an external JavaScript file named `yourfile.js`?
 - A. `<extscript type="text/javascript" src="yourfile.js"></extscript>`
 - B. `<script type="text/javascript" src="yourfile.js"></script>`
 - C. `<script language="yourfile.js"></script>`
 - D. `<script type="text/javascript" link="yourfile.js"></script>`

7. In HTML, the script tag is not case sensitive. However, with XHTML, the script tag must be in _____.
8. The _____ signals the end of a JavaScript statement.
 - A. colon
 - B. period
 - C. question mark
 - D. semicolon
9. To write a string of text on a Web page, the _____ method can be used.
 - A. document.write()
 - B. document.print()
 - C. document.type()
 - D. window.print()
10. When would it be a good idea to use an external JavaScript file?
 - A. When the script is short and going to be used in only one HTML document
 - B. When your Web site viewers have older browsers
 - C. When the script is very long or needs to be placed in more than one HTML document
 - D. External files are not a good idea
11. JavaScript comments can be very useful for the purpose of _____ or _____ your code.
12. Which of the following indicates that a single line of commentary will follow it within JavaScript code?
 - A. /*
 - B. /-
 - C. //
 - D. <!--
13. Which of the following indicates that more than one line of commentary will follow it within JavaScript code?
 - A. /*
 - B. /-
 - C. //
 - D. <!--

- 14.** Which of the following indicates the end of a multiple-line JavaScript comment?
- A.** `\\`
 - B.** `-->`
 - C.** `/*`
 - D.** `*/`
- 15.** When you use multiple-line JavaScript comments, you need to be careful to _____ them.
- A.** close
 - B.** read
 - C.** program
 - D.** compile



Chapter 3

Using Variables