



JavaScript

for Web Warriors

Seventh Edition

Patrick Carey

Sasha Vodnik

JavaScript

for Web Warriors

Seventh Edition

Patrick Carey

Sasha Vodnik



Australia • Brazil • Canada • Mexico • Singapore • United Kingdom • United States

This is an electronic version of the print textbook. Due to electronic rights restrictions, some third party content may be suppressed. Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. The publisher reserves the right to remove content from this title at any time if subsequent rights restrictions require it. For valuable information on pricing, previous editions, changes to current editions, and alternate formats, please visit www.cengage.com/highered to search by ISBN#, author, title, or keyword for materials in your areas of interest.

Important Notice: Media content referenced within the product description or the product text may not be available in the eBook version.

JavaScript for Web Warriors, Seventh Edition
Patrick Carey / Sasha Vodnik

SVP, Higher Education & Skills Product: Erin Joyner

VP, Higher Education & Skills Product: Thais Alencar

Product Director: Mark Santee

Associate Product Manager: Tran Pham

Product Assistant: Tom Benedetto

Learning Designer: Mary Convertino

Senior Content Manager: Michelle Ruelos Cannistraci

Digital Delivery Lead: David O'Connor

Technical Editor: Danielle Shaw

Developmental Editor: Deb Kaufmann

Vice President, Product Marketing: Jason Sakos

Director, Marketing: Danae April

Marketing Manager: Mackenzie Paine

IP Analyst: Ashley Maynard

IP Project Manager: Nick Barrows

Production Service: SPI Global

Designer: Erin Griffin

Cover Image Source: NesaCera/Shutterstock.com

© 2022, 2015 Cengage Learning, Inc.

WCN: 02-300

Unless otherwise noted, all content is © Cengage.

ALL RIGHTS RESERVED. No part of this work covered by the copyright herein may be reproduced or distributed in any form or by any means, except as permitted by U.S. copyright law, without the prior written permission of the copyright owner.

For product information and technology assistance, contact us at
Cengage Customer & Sales Support, 1-800-354-9706
or **support.cengage.com**.

For permission to use material from this text or product, submit all requests online at **www.cengage.com/permissions**.

Library of Congress Control Number: 2021909896

ISBN: 978-0-357-63800-2

Cengage
200 Pier 4 Boulevard
Boston, MA 02210
USA

Cengage is a leading provider of customized learning solutions with employees residing in nearly 40 different countries and sales in more than 125 countries around the world. Find your local representative at **www.cengage.com**.

To learn more about Cengage platforms and services, register or access your online learning solution, or purchase materials for your course, visit **www.cengage.com**.

Notice to the Reader

Publisher does not warrant or guarantee any of the products described herein or perform any independent analysis in connection with any of the product information contained herein. Publisher does not assume, and expressly disclaims, any obligation to obtain and include information other than that provided to it by the manufacturer. The reader is expressly warned to consider and adopt all safety precautions that might be indicated by the activities described herein and to avoid all potential hazards. By following the instructions contained herein, the reader willingly assumes all risks in connection with such instructions. The publisher makes no representations or warranties of any kind, including but not limited to, the warranties of fitness for particular purpose or merchantability, nor are any such representations implied with respect to the material set forth herein, and the publisher takes no responsibility with respect to such material. The publisher shall not be liable for any special, consequential, or exemplary damages resulting, in whole or part, from the readers' use of, or reliance upon, this material.

BRIEF CONTENTS

PREFACE	xi
CHAPTER 1 Introduction to JavaScript	1
CHAPTER 2 Working with Functions, Data Types, and Operators	37
CHAPTER 3 Building Arrays and Controlling Flow	77
CHAPTER 4 Debugging and Error Handling	117
CHAPTER 5 Creating a Web App Using the Document Object Model	163
CHAPTER 6 Enhancing and Validating Forms	209
CHAPTER 7 Manipulating Data in Strings, Arrays, and Other Objects	253
CHAPTER 8 Creating Customized Objects, Properties, and Methods	309
CHAPTER 9 Managing State Information and Security	363
CHAPTER 10 Programming with Event Objects and Third-Party APIs	401
CHAPTER 11 Managing Data Requests with AJAX and Fetch	449
CHAPTER 12 Introducing jQuery	501
APPENDIX A Installing and Configuring a Testing Server	535
APPENDIX B Working with HTML and CSS	543
APPENDIX C Solutions to Quick Checks	Online
GLOSSARY	549
INDEX	561

CONTENTS

PREFACE	xi	Working with Libraries	26
CHAPTER 1		Validating Web Pages	27
INTRODUCTION TO JAVASCRIPT	1	Summary	28
Exploring the JavaScript Language	1	Key Terms	29
Introducing Scripting Languages	2	Review Questions	30
JavaScript and ECMAScript	2	Hands-On Projects	31
The DOM and the BOM	3	Case Projects	35
Understanding Client/Server Architecture	4		
JavaScript and Client-Side Scripting	5	CHAPTER 2	
Understanding Server-Side Scripting	6	WORKING WITH FUNCTIONS, DATA TYPES, AND OPERATORS	37
Should You Use Client-Side or Server-Side Scripting?	6	Working with Functions	38
Writing a JavaScript Program	7	Defining a Function	38
IDEs and Code Editors	7	Writing a Function	39
The <code>script</code> Element	9	Calling a Function	41
JavaScript Statements	9	Returning a Value from a Function	41
Understanding JavaScript Objects	10	Managing Events with Functions	42
Using the <code>write()</code> Method	10	Using Event Handlers	42
Case Sensitivity in JavaScript	13	Events as Object Properties	43
Adding Comments to a JavaScript Program	13	Event Listeners	43
Writing Basic JavaScript Code	14	Events and Anonymous Functions	44
Using Variables	14	Applying a Function to an Event	44
Assigning Variable Names	15	Using Built-in JavaScript Functions	45
Declaring and Initializing Variables	15	Understanding Variable Scope	45
Building Expressions with Variables	17	<code>let</code> and <code>var</code> Declaration Scopes	46
Building an Expression	17	Local and Global Scope	46
Modifying Variables	18	Working with Data Types	48
Understanding Events	18	Working with Numeric Values	48
Working with Elements and Events	19	Working with Boolean Values	49
Referencing Web Page Elements	21	Working with Strings	49
Structuring JavaScript Code	22	Escape Characters and Sequences	50
Including a <code>script</code> Element for Each Code Section	23	Using Operators to Build Expressions	51
Placing the <code>script</code> Element	23	Arithmetic Operators	52
Creating a JavaScript Source File	23	Assignment Operators	53
Referencing an External File	24	Comparison Operators	54
Using the <code>async</code> and <code>defer</code> Keywords	24	Conditional Operators	55
Connecting to a JavaScript File	25	Understanding Falsy and Truthy Values	55
		Logical Operators	56
		Special Operators	57

Understanding Operator Precedence	57	Managing Program Loops and Conditional Statements	105
Using Expressions with Web Form Controls	59	The <code>break</code> Statement	105
Working with Input Control Values	59	The <code>continue</code> Statement	105
Working with Checkboxes	61	Statement Labels	105
Using the <code>change</code> Event with Web Form Controls	64	Summary	107
Locating Errors with the Browser Console	66	Key Terms	107
Accessing the Browser Console	66	Review Questions	108
Locating an Error in Your Program	67	Hands-On Projects	109
Summary	68	Case Projects	116
Key Terms	69	CHAPTER 4	
Review Questions	70	DEBUGGING AND ERROR HANDLING	117
Hands-On Projects	71	Introduction to Debugging	117
Case Projects	76	Load-Time Errors	118
CHAPTER 3		Runtime Errors	119
BUILDING ARRAYS AND CONTROLLING FLOW	77	Logic Errors	120
Storing Data in Arrays	77	Starting Debugging with the Browser Console	121
Declaring and Initializing Arrays	78	Running Javascript in Strict Mode	125
Elements and Indexes	79	Tracing Errors to Their Source	127
Creating an Array	79	Tracing Errors with the <code>window.alert()</code> Method	127
Multidimensional Arrays	82	Tracing Errors with the Console Log	131
Exploring HTML Collections	83	Using Comments to Locate Bugs	135
Referencing an Element within a Collection	83	Tracking Program Flow with Debugging Tools	136
Searching through the DOM	84	Accessing your Browser's Debugging Tools	136
Viewing Arrays and HTML Collections with the Console	85	Adding and Removing Break Points	137
Working with Program Loops	86	Stepping through the Program Execution	140
The <code>while</code> Loop	86	Tracking Variables and Expressions	141
The <code>do while</code> Loop	89	Examining the Call Stack	143
The <code>for</code> Loop	89	Managing Errors	145
Writing a <code>for</code> Loop	91	Handling Exceptions with the <code>try catch</code> Statement	145
Exploring Array Methods for Generating Loops	94	Throwing an Exception	146
Adding Decision Making to Your Code	96	The <code>try catch finally</code> Statement	146
The <code>if</code> Statement	96	The <code>error</code> Parameter in the <code>catch</code> Statement	147
The <code>if else</code> Statement	97	Applying Exception Handling to a Program	147
The <code>else if</code> Statements	97	Customizing Your Error Handling	150
Nested <code>if</code> and <code>if else</code> Statements	100	Catching Errors with the <code>error</code> Event	150
Conditional Statements and Browser Testing	101	Error Handling Functions	151
The <code>switch</code> Statement	102		

Summary	152	Summary	196
Key Terms	153	Key Terms	197
Review Questions	153	Review Questions	197
Hands-On Projects	154	Hands-On Projects	199
Case Projects	160	Case Projects	206
CHAPTER 5		CHAPTER 6	
CREATING A WEB APP USING THE DOCUMENT OBJECT MODEL	163	ENHANCING AND VALIDATING FORMS	209
Designing a Web App	163	Exploring Forms and Form Elements	209
Introducing Nodes	165	The Forms Collection	211
Nodes and the Document Object Model	166	Working with Form Elements	211
Selecting Nodes with the <code>querySelectorAll()</code> Method	168	Properties and Methods of <code>input</code> Elements	212
Creating and Connecting Nodes	168	Navigating Between Input Controls	213
Elements Nodes and HTML Attributes	169	Working with Selection Lists	214
Nodes and Inline Styles	170	Working with Option Buttons	217
Creating a Document Fragment in an App	170	Locating the Checked Option	217
Viewing Elements within the Browser Debugger	173	Accessing the Option Label	220
Restructuring a Node Tree	177	Formatting Data Values in a Form	220
Moving Nodes with the <code>appendChild()</code> Method	177	The <code>toFixed()</code> Method	220
Moving Nodes with the <code>insertBefore()</code> Method	179	Formatting Values Using a Locale String	221
Cloning a Node	181	Responding to Form Events	223
Running Timed Commands	181	Working with Hidden Fields	225
Repeating Commands at Specified Intervals	181	Exploring Form Submission	227
Stopping a Timed Command	182	Using the <code>submit</code> Event	227
Using Time-Delayed Commands	183	Resetting a Form	227
Working with Popup Windows	184	Validating Form Data with JavaScript	228
System Dialog Boxes	184	Working with the Constraint Validation API	230
Working with Browser Windows	185	Exploring the <code>ValidityState</code> Object	231
Writing Content to a Browser Window	187	Creating a Custom Validation Message	232
Limitations of Browser Windows	187	Responding to Invalid Data	233
Creating an Overlay	188	Validating Data with Pattern Matching	235
Introducing the <code>this</code> Object	190	Validating a Selection List	236
Removing a Node	191	Testing a Form Field Against a Regular Expression	238
Exploring the Browser Object Model	193	Creating a Custom Validity Check	240
The <code>History</code> Object	193	Managing Form Validation	241
The <code>location</code> Object	194	Summary	243
The <code>navigator</code> Object	194	Key Terms	244
The <code>screen</code> Object	195	Review Questions	244
		Hands-On Projects	246
		Case Projects	252

CHAPTER 7**MANIPULATING DATA IN STRINGS, ARRAYS, AND OTHER OBJECTS 253****Retrieving Content from a Text File 253**The `file` Object 255

The File Reader API 256

Working with Text Strings 258

Searching for Substrings within a Text String 259

Modifying Text Strings 261

Extracting Characters and Substrings 263

Combining Text Strings 264

Comparing Text Strings 265

Introducing Regular Expressions 266

Matching a Substring 266

Setting Regular Expression Flags 267

Defining Character Types and Character Classes 268

Specifying Repeating Characters 270

Using Escape Sequences 272

Specifying Alternate Patterns and Grouping 273

Programming with Regular Expressions 274

Regular Expression Methods 275

Replacing Text with Regular Expressions 276

Splitting a Text String into an Array 278

Referencing Substring Matches 279

Exploring Array Methods 280

Reversing and Sorting an Array 281

Sorting with a Compare Function 284

Extracting and Inserting Array Items 286

Using Arrays as Data Stacks 287

Exploring the `Math` Object 292The `Math` Object 292`Math` Object Properties 292Applying a `Math` Method to an Array 293

Random Numbers and Random Sorting 294

Exploring the `Date` Object 294

Extracting Information from Dates and Times 295

Setting Date and Time Values 296

Exploring Template Literals 297

Adding Placeholders to Template Literals 297

Tagging a Template Literal 297

Summary 299**Key Terms 300****Review Questions 300****Hands-On Projects 301****Case Projects 308****CHAPTER 8****CREATING CUSTOMIZED OBJECTS, PROPERTIES, AND METHODS 309****Understanding Object-Oriented Programming 309**

Reusing Software Objects 310

Understanding Encapsulation 310

Creating an Object Literal 312

Dot Operators and Bracket Notation 313

Creating a Custom Method 315

Creating an Object with the `new` Operator 317**Working with Object Classes 318**

Understanding Object Classes 318

Object Constructors and Literals 318

Constructor Functions 319

Combining Object Classes 320

Working with Object Prototypes 327

The Prototype Object 327

Extending Built-in JavaScript Objects 329

Introducing Closures 331

Lexical Scope 331

Closures and the Lexical Environment 332

Closures with `for` Loops 334**Working with Public, Private, and Privileged Methods 338****Combining Objects with Prototype Chains 343**

Creating a Prototype Chain 344

Using the Base Object 345

Using the `apply()` and `call()` Methods 346**Data Storage with Associative Arrays 347**The `for in` and `for of` Loops 347

Storing Object data in JSON 349

Summary	351	Hands-On Projects	391
Key Terms	352	Case Projects	399
Review Questions	353		
Hands-On Projects	354		
Case Projects	362		
CHAPTER 9			
<hr/>			
MANAGING STATE INFORMATION AND SECURITY	363	PROGRAMMING WITH EVENT OBJECTS AND THIRD-PARTY APIS	401
Understanding Sessions and State Information	363	Working with Events as Objects	402
Sharing Data Between Forms	365	The <code>Event</code> Object	402
Retrieving the Query String Text using the <code>Location</code> object	367	Event Capturing and Bubbling	403
Replacing URI Encoding Characters	368	Exploring Mouse, Touch, and Pointer Events	406
Introducing Web Storage	371	Exploring Touch Events	407
The Web Storage API	371	Managing Multiple Touchpoints	409
Local Storage and Session Storage Objects	371	Using Pointer Events	409
Storing Data in Web Storage	372	Programming a Drag and Drop Action	410
Viewing Web Storage Items in your Browser	374	Finding Event Coordinates	411
Retrieving Items with the <code>getItem()</code> Method	374	Dragging and Dropping an Element	413
Removing Items from Web Storage	376	Browser Tools for Touchscreen Emulation	415
Exploring Storage Events	377	Exploring the Drag and Drop API	415
Web Storage and the Same-Origin Policy	378	The HTML Drag and Drop API	416
Introducing Cookies	379	Transferring Data with Drag and Drop	417
Cookies vs. Web Storage	379	Working with Keyboard Events	418
The Structure of a Cookie	380	Creating an Interactive Map	421
Writing Data into a Cookie	381	Getting Started with the Google Maps API	421
Setting the Cookie Expiration Date	382	The <code>map</code> Object	423
Setting the Cookie Path	383	Adding Map Pins	425
Setting the Cookie Domain	383	Mapping Your Position with Geolocation	426
Defining Cookie Security	383	Adding Directions to a Map	430
A Function to Write the Cookie Value	384	The <code>route</code> Object	430
Reading a Cookie	385	Displaying the Driving Route	431
Deleting a Cookie	386	Introducing the Device Orientation API	434
Exploring Security Issues	386	Preparing an App for Mobile Use	435
Secure Coding with JavaScript	387	Testing Tools	435
JavaScript Security Concerns	387	Minimizing Download Size	435
Using Third-Party Scripts	388	Minifying Files	435
Summary	389	Summary	437
Key Terms	389	Key Terms	437
Review Questions	390	Review Questions	438
		Hands-On Projects	439
		Case Projects	446

CHAPTER 11**MANAGING DATA REQUESTS WITH AJAX AND FETCH 449**

Introducing Server Requests 450

Exploring HTTP Messages 452

Understanding HTTP Messages 452

Introducing AJAX 454

The XMLHttpRequest Object 454

Managing a Response 456

Viewing the Status of a Request and Response 458

AJAX and Callback Hell 460

Introducing Arrow Functions 461

Arrow Functions and Parameter Values 462

Exploring the Promise Object 464

Defining a Promise Object 464

Chaining Promises 465

Running Multiple Promises 466

Using the Fetch API 467

Managing Fetch Responses 467

Error Handling with Fetch 468

Using Fetch to Return a Search 469

Working with XML 472

Parsing XML Content 473

Working with an XML Node Tree 474

Creating an Autocomplete Search Box 476

Working with JSON Data 477

Building the Suggestion Box 479

Working with Third-Party APIs 482

Requesting a Random GIF 482

Third-Party Endpoints 483

Exploring Security Issues with APIs 486

Working with CORS 486

Using JSONP 486

Using XHR with a Proxy 487

Summary 488**Key Terms 488****Review Questions 489****Hands-On Projects 490****Case Projects 500****CHAPTER 12****INTRODUCING jQuery 501**

Getting Started with jQuery 501

Versions of jQuery 502

Loading jQuery 502

Is jQuery still Relevant? 503

Working with jQuery Selectors 505

Selecting Elements from the DOM 506

Traversing the DOM with jQuery 507

Working with Attributes and CSS Properties 508

Changing the DOM Structure 509

Handling Events with jQuery 511

Working with Effects and Animations 515

Chaining Effects 516

Creating Custom Effects with Animate 517

Controlling the Animation Queue 519

Exploring jQuery Plugins 520

Summary 524**Key Terms 524****Review Questions 525****Hands-On Projects 526****Case Projects 533****APPENDIX A****INSTALLING AND CONFIGURING A TESTING SERVER 535****APPENDIX B****WORKING WITH HTML AND CSS 543****APPENDIX C SOLUTIONS TO QUICK CHECKS ONLINE****GLOSSARY 549****INDEX 561**



JavaScript is a client-side scripting language that allows web page authors to develop interactive web pages and sites. Although JavaScript is considered a programming language, it is also a critical part of web page design and authoring. This is because the JavaScript language enables web developers to add functionality directly to a web page's elements. The language is relatively easy to learn, allowing non-programmers to quickly incorporate JavaScript functionality into a web page. In fact, because it is used extensively in the countless web pages that are available on the World Wide Web, JavaScript is arguably the most widely used programming language in the world.

JavaScript, Seventh Edition, teaches web page development with JavaScript for students with little programming experience. Although it starts with an overview of the components of web page development, students using this book should have basic knowledge of web page creation, including familiarity with commonly used HTML elements and CSS properties. This book covers the basics of ECMAScript Edition 11 (June, 2020), which is supported by all modern browsers. This book also covers advanced topics including object-oriented programming, the Document Object Model (DOM), touch and mobile interfaces, and Fetch. The HTML documents in this book are written to HTML5 standards, with some XHTML-compatible element syntax. After completing this course, you will be able to use JavaScript to build professional quality web applications.

The Approach

This book introduces a variety of techniques, focusing on what you need to know to start writing JavaScript programs. In each chapter, you perform tasks that let you use a particular technique to build JavaScript programs. The step-by-step tasks are guided activities that reinforce the skills you learn in the chapter and build on your learning experience by providing additional ways to apply your knowledge in new situations. In addition to step-by-step tasks, each chapter includes objectives, short quizzes, a summary, key terms with definitions, review questions, and reinforcement exercises that highlight major concepts and let you practice the techniques you've learned.

Course Overview

The examples and exercises in this book will help you achieve the following objectives:

- › Use JavaScript with HTML elements
- › Work with JavaScript variables and data types and learn how to use the operations that can be performed on them
- › Add functions and control flow within your JavaScript programs
- › Trace and resolve errors in JavaScript programs
- › Write JavaScript code that controls the web browser through the browser object model
- › Use JavaScript to make sure data was entered properly into form fields and to perform other types of preprocessing before form data is sent to a server
- › Create JavaScript applications that use object-oriented programming techniques
- › Manipulate data in strings and arrays
- › Save state information using hidden form fields, query strings, cookies, and Web Storage
- › Incorporate touchscreen support and mobile capabilities in web applications
- › Dynamically update web applications with Ajax and Fetch
- › Build a web application using the jQuery library

JavaScript, Seventh Edition, presents twelve chapters that cover specific aspects of JavaScript programming. **Chapter 1** discusses basic concepts of the World Wide Web, reviews HTML documents, and covers the basics of how to add JavaScript to web pages. How to write basic JavaScript code, including how to use variables, data types, expressions, operators, and events, is also discussed in Chapter 1. This early introduction of key JavaScript concepts gives you a framework for better understanding more advanced concepts and techniques later in this book, and allows you to work on more comprehensive projects from the start. **Chapter 2** covers functions, data types, and how to build expressions. **Chapter 3** explains how to store data in arrays and how to use structured logic in control structures and statements. **Chapter 4** provides a thorough discussion of debugging techniques, including how to use the browser consoles integrated into all modern browsers. **Chapter 5** teaches how to manipulate the structure of a web document by creating element nodes and web page overlays. **Chapter 6** explains how to use JavaScript to make sure data was entered properly into form fields and how to perform other types of preprocessing before form data is sent to a server. **Chapter 7** covers advanced topics in manipulating data in text strings, arrays, and JSON. **Chapter 8** presents object-oriented programming concepts, including coverage of object classes and closures. **Chapter 9** explains how to save state information using hidden form fields, query strings, cookies, and Web Storage, and also briefly discusses JavaScript security issues. **Chapter 10** covers supporting touch and pointer events in a web application, as well as using data provided by mobile device hardware and optimizing a web app for mobile users. **Chapter 11** introduces the basics of how to use Ajax and Fetch to dynamically update portions of a web page with server-side data. **Chapter 12** introduces using the jQuery library to simplify common programming tasks in JavaScript. **Appendix A** provides detailed instructions on installing the XAMPP web server on a local machine. **Appendix B** gives a brief refresher on the basics of HTML, XHTML, and CSS. **Appendix C**, which is online, lists answers for all Quick Checks.

What's New in This Edition?

The seventh edition includes the following important new features:

- › New coverage of JavaScript topics from ES6 including the `let` and `const` keywords, template literals, and arrow function syntax.
- › Expanded coverage of important programming topics including regular expressions, multidimensional arrays, closures, function expressions, array functions, and sorting callback functions.
- › Expanded coverage of object-oriented programming techniques, including the creation of object classes, object prototypes, and prototype chains.
- › New and expanded coverage of the Event model, event bubbling and capturing, event objects, pointer events, keyboard events, and the Drag and Drop API.
- › New coverage of the Fetch API and JavaScript promises.
- › Expanded coverage of jQuery coding techniques and using the jQuery UI library.
- › Twelve new chapter cases with code written to the latest JavaScript standards and covering such tasks as creating a Lightbox Slideshow, developing an interactive Poker Game, using JavaScript string methods to create a Word Cloud app, creating an interactive route map with the Google Maps API, and retrieving newsfeed data for an online blog.
- › Four new case projects with each chapter and a fifth debugging project that tests the student's ability to locate and fix programming errors.
- › Expanded coverage of browser developer tools for debugging and managing network connections and data.
- › Updated page design makes it easier to follow steps and locate important information to use as a study guide or reference book.

Features

Each chapter in *JavaScript, Seventh Edition*, includes the following features:

- › **Chapter Objectives:** Each chapter begins with a list of the important concepts presented in the chapter. This list provides you with a quick reference to the contents of the chapter as well as a useful study aid.

- › **Figures and Tables:** Plentiful full-color screenshots allow you to check your screen after each change. Tables consolidate important material for easy reference.
- › **Code Examples:** Numerous code examples throughout each chapter are presented in any easy-to-read font.
- › **Key Terms:** The first use of key terms are printed in bold and orange font to draw your attention to important definitions.

Note | These elements provide additional helpful information on specific techniques and concepts.

Common Mistakes

These notes highlight common mistakes that a new programmer might make with the tasks and concepts introduced in the chapter and provide suggestions for locating and fixing those errors.

Skills at Work

These boxes provide guidance for navigating the world of work.

Best Practices

These boxes highlight guidelines for real- world implementation of various topics.

Programming Concepts

These boxes explain principles underlying the subject of each chapter or section.

- › **Quick Check:** Several Quick Checks are included in each chapter. These Quick Checks, consisting of two to five questions, help ensure you understand the major points introduced in the chapter. Appendix C (provided online) gives answers to each chapter's Quick Check questions.
- › **Summary:** These brief overviews revisit the ideas covered in each chapter, providing you with a helpful study guide.
- › **Key Terms List:** These lists compile all new terms introduced in the chapter, creating a convenient reference covering a chapter's important concepts.
- › **Review Questions:** At the end of each chapter, a set of twenty review questions reinforces the main ideas introduced in the chapter. These questions help you determine whether you have mastered the concepts presented in the chapter.
- › **Hands-On Projects:** Although it is important to understand the concepts behind every technology, no amount of theory can improve on real-world experience. To this end, each chapter includes four detailed Hands-On Projects that provide you with practice implementing technology skills in real-world situations. Each project is a standalone project, giving you a wide variety of topics and difficulty levels.

- › **Debugging Challenge:** Each chapter includes one Debugging Challenge project in which you are given code that contains errors preventing it from running or running correctly. Here you can practice the important skill of interpreting other people’s code and repairing it.
- › **Case Projects:** These end-of-chapter projects are designed to help you apply what you have learned to open-ended situations, both individually and as a member of a team. They give you the opportunity to independently synthesize and evaluate information, examine potential solutions, and make decisions about the best way to solve a problem.

MindTap

In addition to the readings, the MindTap includes the following:

- › **Course Orientation:** Custom videos and readings prepare students for the material and coding experiences they will encounter in their course.
- › **Coding Snippets:** These short, ungraded coding activities are embedded in the MindTap Reader and provide students an opportunity to practice new programming concepts “in-the-moment”. The coding Snippets help transition the student from conceptual understanding to application of JavaScript code.

Instructor and Student Resources

Additional instructor and student resources for this product are available online. Instructor assets include an Instructor’s Manual, Solutions and Answer Guide, Solutions Files, Teaching Online Guide, PowerPoint® slides, and a test bank powered by Cengage®. Student assets include data sets for the Hands-On Projects and Case Projects. Sign up or sign in at www.cengage.com to search for and access this product and its online resources.

Read This Before You Begin

The following information will help you prepare to use this textbook.

Data Files

To complete the steps, exercises, and projects in this book, you will need data files that have been created specifically for this book. The data files are available in the Student Resources. Note that you can use a computer in your school lab or your own computer to complete the steps, exercises, and projects in this book.

Using Your Own Computer

You can use a computer in your school lab or your own computer to complete the chapters. To use your own computer, you will need the following:

- › **A modern web browser**, including the current versions of Chrome, Edge, Firefox, or Safari.
- › **A code-based HTML editor**, such as Aptana Studio, Visual Studio Code, Notepad++, Eclipse, Adobe Dreamweaver, or Atom.
- › **A web server** (for Chapter 11) such as Apache HTTP Server or Microsoft Internet Information Services and PHP. Appendix A contains instructions on how to install a web server and PHP.

Acknowledgements

Creating the Seventh Edition of JavaScript has truly been a team effort. Special thanks to Michelle Ruelos Cannistraci, Mary Convertino, Tran Pham, Erin Griffin, and Troy Dundas at Cengage, to developmental editor Deb Kaufmann, and to quality assurance and technical editor Danielle Shaw. Thanks also to the production team of copyeditors, proofreaders, and compositors at SPi Global.

And many thanks to the reviewers who provided valuable feedback: Thomas Brown, Forsyth Technical Community College; Tonya Melvin Bryant, Coastal Carolina University; and Pranshu Gupta, DeSales University.

(Patrick): This book is dedicated to my special girls: Abbey, Nicola, Sonia, Catherine, and most of all, Joan.

Introduction to JavaScript

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

- › Explain the history of JavaScript and scripting languages and how each has been developed for its current use
- › Write content into a web page using JavaScript
- › Add JavaScript code to a web page
- › Create and apply JavaScript variables
- › Work with event handlers within a web page
- › Connect to an external JavaScript File

JavaScript is a programming language that adds complex interactive features to a website. Among its many applications, JavaScript can be used to validate data on web forms, generate new content in response to user actions, and store data that will persist from one web session to the next. JavaScript is an increasingly important tool for the website designer and programmer to create useful and powerful web applications.

This chapter introduces the basics of JavaScript and its role in developing interactive websites. You will create a JavaScript program for use in a web page and explore browser tools for evaluating your code.

Exploring the JavaScript Language

Before discussing the details of JavaScript, this chapter will examine how JavaScript fits in with the development of the web as the primary source of sharing content and commerce across the globe. JavaScript had its origins in the mid-1990s with the creation of the **World Wide Web** or **web**, which was developed to share data across a network of linked documents. In its early years, the web was primarily used for academic research and did not require much more than the ability to share text and graphic images between researchers.

The business world quickly recognized that the web could be a powerful tool for online commerce including the process of validating customer data. When JavaScript first appeared in 1995, it was used to handle as much of that validation as possible to speed up customer transactions. But what is JavaScript and how does it compare to other languages?

Introducing Scripting Languages

In discussing computer languages, especially those associated with website design, this book focuses on three general types of languages: programming languages, scripting languages, and markup languages.

A **programming language** is a set of instructions directing the actions of the computer or computer device. Before these instructions can be performed, they need to be **compiled**, a process by which those instructions are transformed into **machine code** that can be understood by the computer or computer device. The compiling is done by a program called a **compiler**. Thus, before you can work with a programming language, you need to have a working environment to build the code, test the code, and compile it. Examples of programming language include Java, C, C++, and C#. The browser that interacts with the web was created and compiled using a programming language like C++. This book will not examine those languages except in terms of how they might interact with JavaScript.

A **scripting language** belongs to a subcategory of programming languages that do not require compiling but instead are run directly from a program or **script**. Scripting languages need to be **interpreted**, in which the code is read line-by-line by an **interpreter** that scans the code for errors even as it runs. A JavaScript interpreter is built into every web browser, so to create a JavaScript program you only need a text editor to write the code and a web browser to run it. Examples of scripting languages include JavaScript, PHP, Perl, and Python.

Finally, a **markup language** is a language that defines the content, structure, and appearance of a document. Common markup languages include **HTML (Hypertext Markup Language)** used to define the content and structure of your web page and **CSS (Cascading Style Sheets)** used to define how that web page will appear on a specified device. This book focuses on the connections between HTML and CSS, which define the content and appearance of your web pages, and JavaScript, which provides tools for interacting with those pages (see **Figure 1-1**). These chapters assume that you already possess a basic knowledge of HTML and CSS.

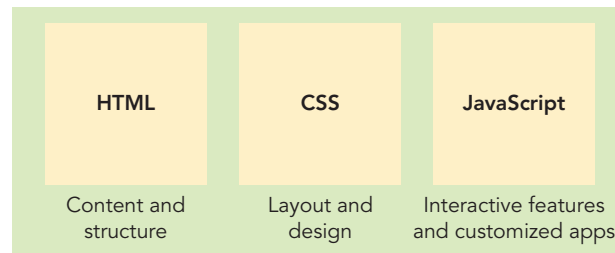


Figure 1-1 The roles of HTML, CSS, and JavaScript in web development

JavaScript and ECMAScript

The version of JavaScript discussed in this book is not the same as the one introduced in 1995. Over the years the scope and power of the language has grown to meet the needs of an ever-changing market that includes an increasing variety of devices from desktop computers to mobile phones. Who determines what JavaScript is and how it will develop is an important part of its story.

In the beginning, JavaScript was developed for the Netscape browser by the Netscape developer Brendan Eich. Shortly thereafter, JavaScript was supported by Microsoft's Internet Explorer browser in a slightly different form called **JScript**. One major headache for developers in the late 1990s was reconciling the differences between JavaScript and JScript as well as keeping up with the changes to the language as each browser sought to add features and tools the other browser lacked. Unlike a programming language such as C, at the time there was no single set of governing standards for JavaScript. Its growth was as unpredictable as the web itself.

Therefore in 1997, JavaScript was submitted to the **European Computer Manufacturers Association (ECMA)** as a proposal for a standardized scripting language that would work across a wide range of devices and browsers. A technical committee composed of developers from the major browser manufacturers was tasked with the goal of developing

a set of standards for the language. The specification for this scripting language is called **ECMAScript** or **ECMA-262**. JavaScript is just one implementation of the ECMAScript standard, but it is the most important.

Every year a different version or edition of ECMAScript is released. Within a few years of release, most browsers will implement the changes in that edition, so while web programmers need to keep apprised of the changes in the most recent ECMAScript edition, they also need to write their code to conform to current browsers *and* older browser versions. **Figure 1-2** describes the most recent editions of ECMAScript at the time of this writing.

ECMAScript Edition	Date Issued	Features
6 th Edition (ES6)	June 2015	Added new syntax for complex applications, included iterators and for . . . of loops, arrow functions, variable declarations using let and const
7 th Edition (ES7)	June 2016	Added block-scoping of variables, exponentiation operator, and support for asynchronous execution
8 th Edition (ES8)	June 2017	Added support for async/await constructions
9 th Edition (ES9)	June 2018	Included rest/spread operators for variables, asynchronous iteration, and additions to regular expressions
10 th Edition (ES10)	June 2019	Added features to object prototypes and changes to Array sorting
11 th Edition (ES11)	June 2020	Added an optional object chaining operator for array and functions

Figure 1-2 Editions of ECMAScript

Note

You can do a search on the web for the current support of different ECMAScript editions by desktop and mobile browsers.

The DOM and the BOM

Though they are often talked about as being identical, JavaScript is more than just ECMAScript. ECMAScript is the scripting language, but it does not tell you how to interact with the contents of a website or the browser. The full implementation of JavaScript is built on three foundations:

- › ECMAScript, which is the core of the programming language, providing the syntax, keywords, properties, methods, and general structure for writing code.
- › The **Document Object Model (DOM)**, which describes how to access the contents of the web page and user actions within that page.
- › The **Browser Object Model (BOM)**, which describes how to access the features and behaviors of the browser itself.

The Document Object Model and the Browser Object Model are examples of an **Application Programming Interface (API)**, which is a set of procedures that access an application such as a web page or a web browser. Just as the specifications for ECMAScript have developed and changed through the years, the specifications for the DOM and the BOM have also grown in response to the need for a robust and powerful scripting language for the web.

The specifications for the DOM are managed by the **World Wide Web Consortium (W3C)**, the same group managing the development of HTML and CSS. **Figure 1-3** describes the different versions of the DOM that have been released over the years. Note that the DOM is used by programming languages other than JavaScript.

Unlike the Document Object Model, there is no formal set of standards for the Browser Object Model. Each browser is different and implements its own version of the BOM, but the BOM is largely the same from one browser to the next because it is to everyone's advantage to adhere to a common standard.

Now that you have had a short overview of JavaScript and its history, let's turn to how JavaScript works with your computer or mobile device and the computers that host the sites on the web you frequently visit.

DOM	DATE	FEATURES
DOM Level 0	1995	Provided a basic interface to access the contents of a web page using the initial version of JavaScript
DOM Level 1	October 1998	Added a way of mapping the content of a web page to JavaScript keywords, functions, properties, and methods
DOM Level 2	December 2000	Added an interface to events occurring within the web page, the contents of CSS style sheets, and the ability to transverse and manipulate the hierarchical structure of the web page content
DOM Level 3	April 2004	Added support for methods to load and save web documents, validate web forms, and provides the ability to work with keyboard objects and events
DOM Level 4	November 2015	An ongoing “living standard” that is updated to reflect the events and actions occurring within the document model based on the evolving needs of the market and mobile devices

Figure 1-3 Versions of the Document Object Model

Understanding Client/Server Architecture

To be successful in web development, you need to understand the basics of client/server architecture. There are many definitions of the terms “client” and “server”. In traditional client/server architecture, the **server** is a device or application from which a client requests information. A server fulfills a request for information by managing the request or serving the requested information to the client—hence the term, “client/server.” A system consisting of a client and a server is known as a **two-tier system**.

One of the primary roles of the **client**, or **front end**, in a two-tier system is the presentation of an interface to the user. The user interface gathers information from the user, submits it to a server, or **back end**, then receives, formats, and presents the results returned from the server. The main responsibilities of a server are usually data storage, management, and communicating with external services. On client/server systems, heavy processing, such as calculations, usually takes place on the server. As devices that are used to access web pages—such as computers, tablets, and mobile phones—have become increasingly powerful, however, many client/server systems have placed increasing amounts of the processing responsibilities on the client. In a typical client/server system, a client computer might contain a front end that is used for requesting information from a database on a server. The server locates records that meet the client request, performs some sort of processing, such as calculations on the data, and then returns the information to the client. The client computer can also perform some processing, such as building the queries that are sent to the server or formatting and presenting the returned data. **Figure 1-4** illustrates the design of a two-tier client/server system.

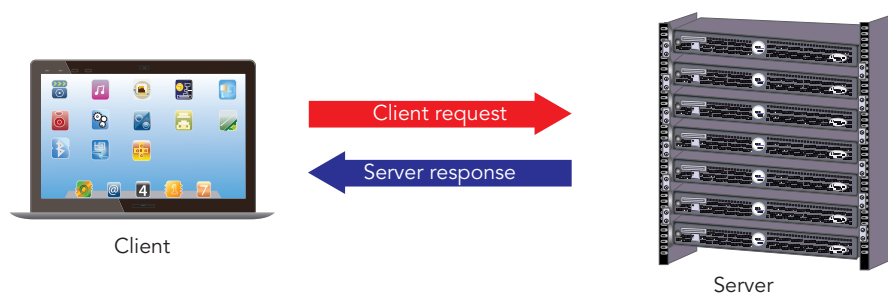


Figure 1-4 A two-tier client/server system

The web is built on a two-tier client/server system, in which a web browser (the client) requests documents from a web server. The web browser is the client user interface. You can think of the web server as a repository for web pages. After a web server returns the requested document, the web browser (as the client user interface) is responsible for formatting and presenting the document to the user. The requests and responses through which a web browser and web server communicate occur via **Hypertext Transfer Protocol (HTTP)**, which is the main system used on the web for exchanging data. For example, if a web browser requests the URL `http://www.cengage.com`, the request is made with HTTP because the URL specifies the HTTP protocol. The web server then returns to the web browser an HTTP response containing the response header and the HTML for the Cengage home page.

After you start adding databases and other types of applications to a web server, the client/ server system evolves into what is known as a three-tier client architecture. A **three-tier client/server system**—also known as a **multitier client/server system** or ***n*-tier client/server system**—consists of three distinct pieces: the client tier, the processing tier, and the data storage tier. The client tier, or user interface tier, is still the web browser. However, the database portion of the two-tier client/server system is split into a processing tier and the data storage tier. The **processing tier**, or **middle tier**, handles the interaction between the web browser client and the data storage tier. (The processing tier is also sometimes called the processing bridge.) Essentially, the client tier makes a request of a database on a web server. The processing tier performs any necessary processing or calculations based on the request from the client tier, and then reads information from or writes information to the data storage tier. The processing tier also handles the return of any information to the client tier. Note that the processing tier is not the only place where processing can occur. The web browser (client tier) still renders web page documents (which requires processing), and the database or application in the data storage tier might also perform some processing.

Note

Two-tier client/server architecture is a physical arrangement in which the client and server are two separate computers. Three-tier client/server architecture is more conceptual than physical, because the storage tier can be located on the same server.

Figure 1-5 illustrates the design of a three-tier client/server system.

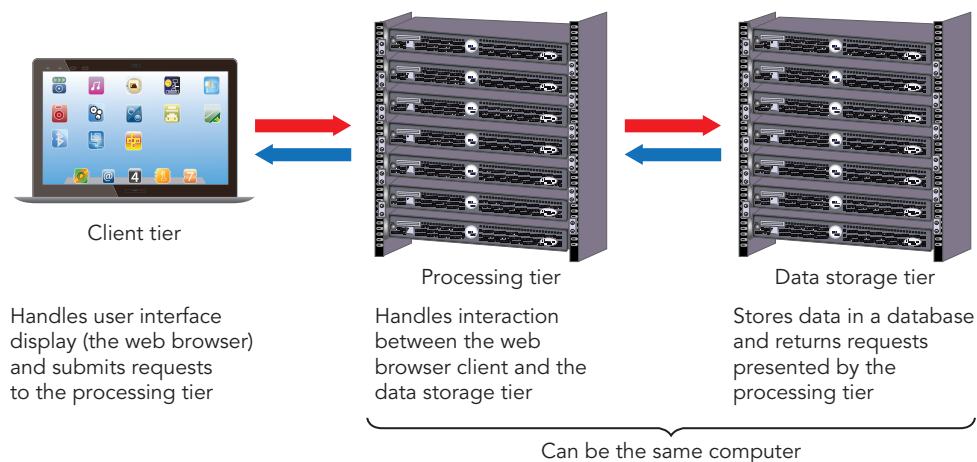


Figure 1-5 A three-tier client/server system

JavaScript and Client-Side Scripting

HTML was not originally intended to control the appearance of pages in a web browser. When HTML was first developed, web pages were **static**—that is, they couldn't change after the browser rendered them. However, after the web grew beyond a small academic and scientific community, people began to recognize that greater interactivity and better visual design would make the web more useful. As commercial applications of the web grew, the demand for more interactive and visually appealing websites also grew.

HTML could be used to produce only static documents. You can think of a static web page written in HTML as being approximately equivalent to a printed book; you can read it or move around in it, but the content is fixed.

What JavaScript provides that HTML needed is **client-side scripting** in which the scripting language runs on a local browser (on the client tier) instead of on a web server (on the processing tier). In this way, web pages can respond dynamically to user actions without putting extra strain on the operations of the server.

Note

Many people think that JavaScript is a simplified version of the Java programming language, or is related to Java in some other way. However, the languages are entirely different. Java is an advanced programming language that was created by Sun Microsystems and is considerably more difficult to master than JavaScript. Although Java can be used to create programs that can run from a web page, Java programs are usually external programs that execute independently of a browser. In contrast, JavaScript programs always run within a web page and control the browser.

For security reasons, the JavaScript programming language cannot be used outside of specific environments. The most common environment where JavaScript is run is a web browser. For example, to prevent malicious scripts from stealing information, such as your email address or the credit card information you use for an online transaction, or from causing damage by changing or deleting files, JavaScript allows manipulation only of select files associated with the browser, and then with strict limitations. Another helpful limitation is the fact that JavaScript cannot run system commands or execute programs on a client. The ability to read and write cookies and a few other types of browser storage is the only type of access to a client that JavaScript has. Web browsers, however, strictly govern their storage and do not allow access to stored information from outside the domain that created it. This security also means that you cannot use JavaScript to interact directly with web servers that operate at the processing tier. Although programmers can employ a few tricks (such as forms and query strings) to allow JavaScript to interact indirectly with a web server, if you want true control over what's happening on the server, you need to use a server-side scripting language.

Understanding Server-Side Scripting

Server-side scripting refers to programming using a scripting language that is executed from a web server. Some of the more popular server-side scripting languages are PHP, ASP.NET, Python, and Ruby. One of the primary reasons for using a server-side scripting language is to develop an interactive website that communicates with a database. Server-side scripting languages work in the processing tier and have the ability to handle communication between the client tier and the data storage tier. At the processing tier, a server-side scripting language usually prepares and processes the data in some way before submitting it to the data storage tier. Some of the more common uses of server-side scripting languages include shopping carts, search engines, discussion forums, and multiplayer games.

Without JavaScript, a server-side scripting language can't access or manipulate the user's web browser. In fact, a server-side scripting language cannot run on a client tier at all. Instead, a server-side scripting language exists and executes solely on a web server, where it performs various types of processing or accesses databases. When a client requests a server-side script, the script is interpreted and executed by the scripting engine within the web server software. After the script finishes executing, the web server software translates the results of the script (such as the result of a calculation or the records returned from a database) into HTML, which it then returns to the client. In other words, a client will never see the serverside script, only the HTML that the web server software returns from the script. **Figure 1-6** illustrates how a web server processes a server-side script.

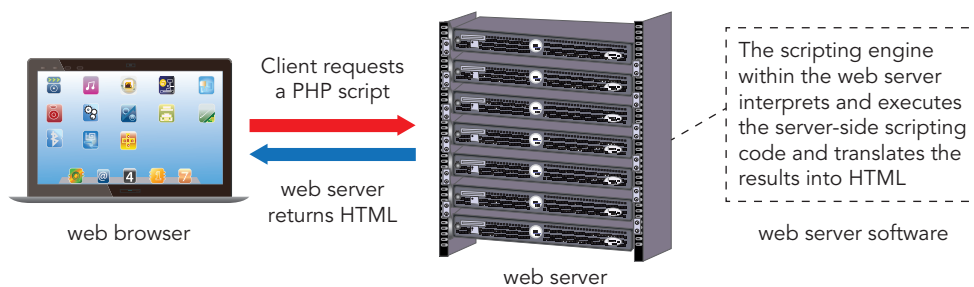


Figure 1-6 How a web server processes a server-side script

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

An important question in the design of any client/server system is deciding how much processing to place on the client and how much to place on the server. In the context of website development, you must decide whether to use client-side JavaScript or a server-side script. This is an important consideration that can greatly affect the performance of

your program. In some cases, the decision is simple. If you want to control the web browser, you must use JavaScript. If you want to access a database on a web server, you must use a server-side script. However, there are tasks that both languages can accomplish, such as validating forms and manipulating cookies. Furthermore, both languages can perform the same types of calculations and data processing.

A general rule of thumb is to allow the client to handle the user interface processing and light processing, such as data validation, but have the web server perform intensive calculations and data storage. This division of labor is especially important when dealing with clients and servers over the web. Unlike with clients on a private network, it's not possible to know in advance the computing capabilities of each client on the web. You cannot assume that each client (browser) that accesses your client/server application (website) has the necessary power to perform the processing required by an application. For this reason, intensive processing should be performed on the server.

Because servers are usually much more powerful than client computers, your first instinct might be to let the server handle all processing and only use the client to display a user interface. Although you do not want to overwhelm clients with processing they cannot handle, it is important to perform as much processing as possible on the client for several reasons:

- › Distributing processing among multiple clients creates applications that are more powerful, because the processing power is not limited to the capabilities of a single computer. Client devices—including computers, tablets, and smartphones—become more powerful every day. Thus, it makes sense to use a web application to harness some of this power and capability. A **web application** is a program that is executed on a server but is accessed through a web page loaded in a client browser.
- › Local processing on client computers minimizes transfer times across the Internet and creates faster applications. If a client had to wait for all processing to be performed on the server, a web application could be painfully slow over a low-bandwidth Internet connection.
- › Performing processing on client computers decreases the amount of server resources needed by application providers, decreasing costs for infrastructure and power use.

Now that you have seen how JavaScript fits within the client/server structure, in the next section you will explore how to start applying JavaScript to your own web pages.

Quick Check 1

1. How does a scripting language like JavaScript differ from a programming language like C#?
2. What are the three core foundations upon which JavaScript is built?
3. In client/server architecture, what is a client? What is a server?

Writing a JavaScript Program

Before you start writing JavaScript you must first choose an application in which to create your programs. You can work with IDEs or code editors.

IDEs and Code Editors

You have a lot of choices for creating your own JavaScript programs. Like HTML and CSS, writing JavaScript code requires only a basic text editor but you can also use an **Integrated Development Environment (IDE)** to manage all of the facets of website development, including the writing and testing of JavaScript code. Popular IDEs include the following:

- › Microsoft Visual Studio (<https://visualstudio.microsoft.com>)
- › Komodo IDE (<https://www.activestate.com/products/komodo-ide>)
- › Aptana Studio (<http://www.aptana.com>)
- › NetBeans (<https://netbeans.org>)

If you find an IDE to be either too expensive (though there are very good free IDEs available on the web) or containing too much overhead for your projects, you might be better suited with a **code editor** that simply manages the writing of HTML, CSS, and JavaScript code within a graphical interface. These editors include a number of features that make coding easier, including numbering the lines of code in a document and color coding text based on meaning—for instance, displaying JavaScript keywords in one color and user-defined text and values in another. Several good free code editors are available online, including the following:

- › Visual Studio Code (<https://code.visualstudio.com>)
- › Notepad++ (<https://notepad-plus-plus.org>)
- › Brackets (<http://brackets.io>)
- › Atom (<https://atom.io>)

The HTML, CSS, and JavaScript code samples displayed in this book are based on a code editor that uses color to distinguish different parts of the code. Your code editor might use a different coloring scheme, but that will not affect the code because HTML, CSS, and JavaScript are saved as basic text.

In this chapter, you will add JavaScript code to a web page for Tinley Xeriscapes, a landscaping company that specializes in plants that need minimal watering. A designer has created a new layout for the company's website, and they would like you to incorporate JavaScript to enhance the functionality of one of the site's pages. **Figure 1-7** shows a preview of the completed web page incorporating the functionality you will create in this chapter.

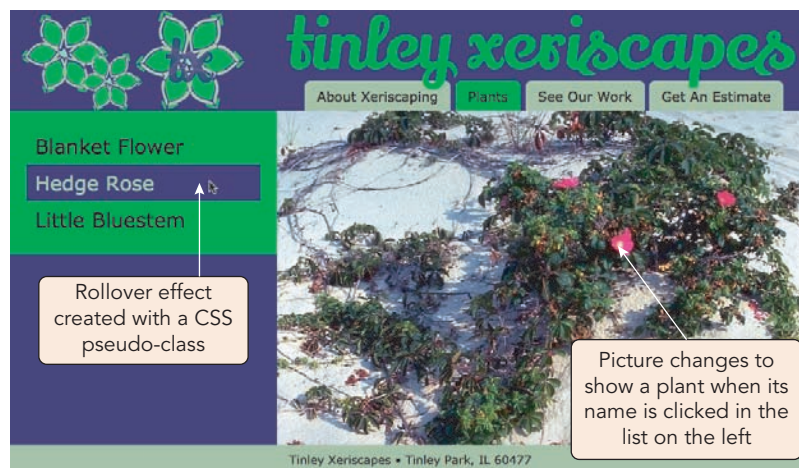


Figure 1-7 Completed Tinley Xeriscapes Plants page using JavaScript

U.S. Department of Agriculture

Open the HTML file for this web page now.

To open the Tinley Xeriscapes page:

1. Use your code editor to go to the js01 ► chapter folder of your data files.
2. Open the **js01_txt.html** file in your code editor.
3. Within the head section of the HTML file, enter your name and the date in the Author and Date lines.
4. Save the file as **js01.html**.

Next, begin writing the code of your first JavaScript program.

The `script` Element

JavaScript can be added to a web page by embedding the code within the following `script` element:

```
<script>
    statements
</script>
```

where *statements* are the individual lines of code in the JavaScript program. The following `script` element contains a single JavaScript statement displaying an alert window with the text message “Hello World”:

```
<script>
    window.alert("Hello World");
</script>
```

When the browser encounters a `script` element, it stops loading the page and processes the statements enclosed within the script. In this case, the browser would stop loading the page to display the “Hello World” message. Once the script is run, the browser continues to process the remaining content in the HTML file. Add a `script` element now within the opening and closing `figcaption` tags in the HTML file for the Tinley Xeriscapes page.

To add the `script` element to the page:

1. Scroll down to the `article` element in the `js01.html` file within your code editor.
2. After the opening `figcaption` tag, type:

```
<script>
</script>
```

indenting the opening and closing tags to make your code easier to read. See **Figure 1-8**.

```
<article>
  <figure>
    
    <figcaption id="imgCaption">
      <script>
      </script>
    </figcaption>
  </figure>
</article>
```

The script element encloses JavaScript code within an HTML file

Figure 1-8 Adding a `script` element

3. Save your changes to the file.

Next you will learn general rules for writing statements in JavaScript.

JavaScript Statements

The individual lines of code, or **statements**, that make up a JavaScript program in a document are contained within the `script` element. The following script contains a single statement that writes the text “Plant choices” to a web browser window, using the `write()` method of the `Document` object, which you will study shortly:

```
document.write("<p>Plant choices</p>");
```

Notice that the preceding statement ends in a semicolon. Many programming languages, including C++ and Java, require you to end all statements with a semicolon. JavaScript statements are not required to end in semicolons. Semicolons are strictly necessary only when you want to separate statements that are written on a single line. However, it is considered good JavaScript programming practice to end every statement with a semicolon whether strictly required or not. This is the convention that will be used in this book.