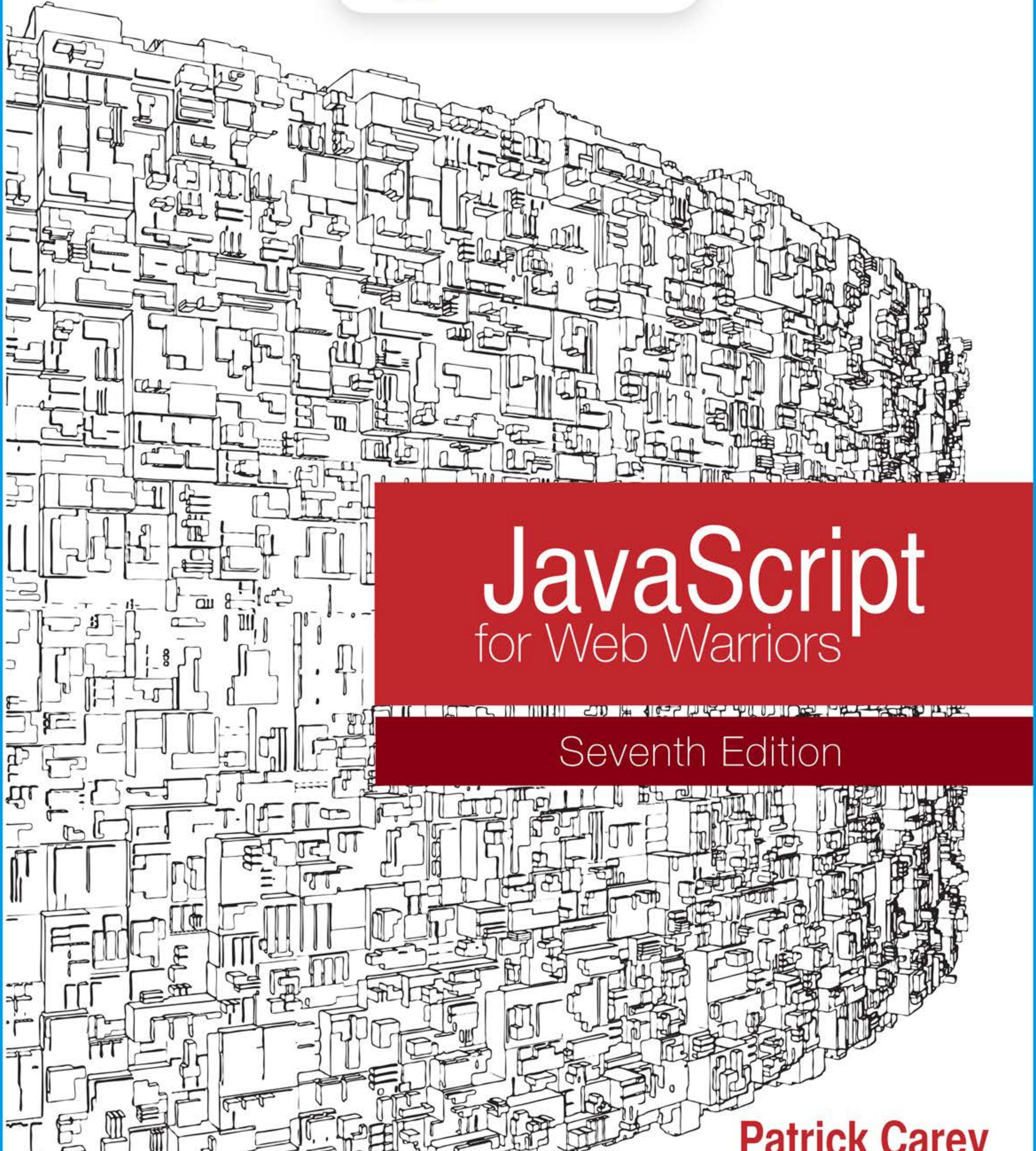
CENGAGE



JavaScript for Web Warriors

Seventh Edition

Patrick Carey

Sasha Vodnik

JavaScript for Web Warriors

Seventh Edition

Patrick Carey

Sasha Vodnik

CENGAGE

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

CENGAGE

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

Library of Congress Control Number: 2021909896

ISBN: 978-0-357-63800-2

Cengage
200 Pier 4 Boulevard
Boston, MA 02210
USA

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.

Printed in the United States of America
Print Number: 01 Print Year: 2021

BRIEF CONTENTS

CONTENTS

PREFACE

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 Cognero®. 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.

CHAPTER 1

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.

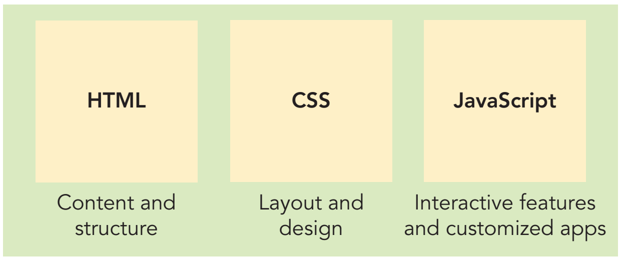


Figure 1-1 The roles of HTML, CSS, and JavaScript in web development

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

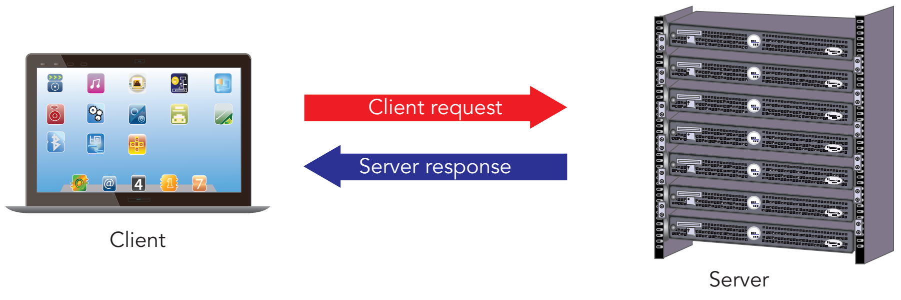


Figure 1-4 A two-tier client/server system

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.

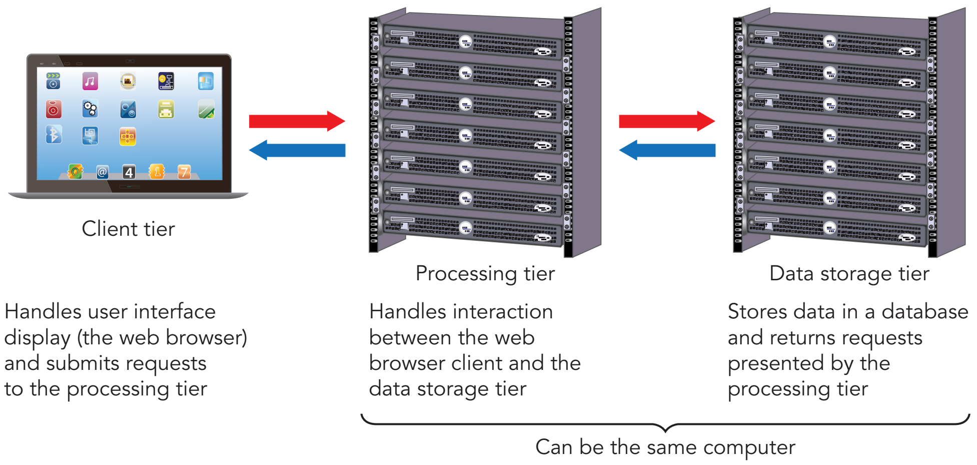

Client tier
Processing tier
Data storage tier
Handles user interface display (the web browser) and submits requests to the processing tier
Handles interaction between the web browser client and the data storage tier
Stores data in a database and returns requests presented by the processing tier
Can be the same computer

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.

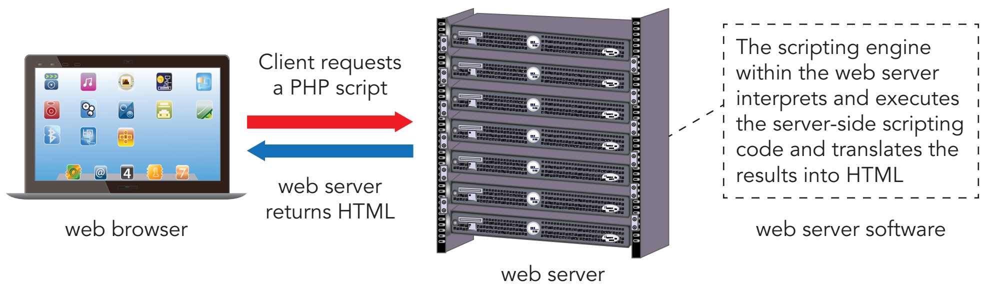

Client requests a PHP script
web server returns HTML
The scripting engine within the web server interprets and executes the server-side scripting code and translates the results into HTML
web browser
web server
web server software

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.

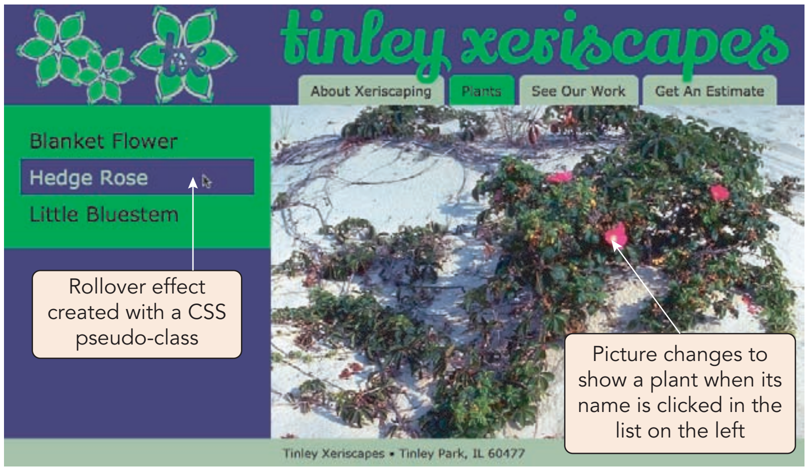

tinley xeriscapes
About Xeriscaping
Plants
See Our Work
Get An Estimate
Blanket Flower
Hedge Rose
Little Bluestem
Rollover effect created with a CSS pseudo-class
Picture changes to show a plant when its name is clicked in the list on the left
Tinley Xeriscapes • Tinley Park, IL 60477

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>
```

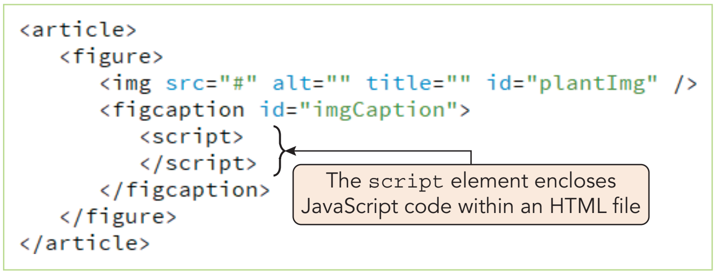
When the browser encounters a `script` element, it stops loading the page and processes the statements enclosed within the script. In this case, the browser would stop loading the page to display the "Hello World" message. Once the script is run, the browser continues to process the remaining content 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>
      <img src="#" alt="" title="" id="plantImg" />
      <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.