TRAINING & REFERENCE

murach's HTML and CSS

5TH EDITION

Zak Ruvalcaba
Anne Boehm

MIKE MURACH & ASSOCIATES, INC.
3730 W Swift Ave. • Fresno, CA 93722
www.murach.com • murachbooks@murach.com

Editorial team

Authors: Zak Ruvalcaba
Anne Boehm

Editor: Mike Murach

Production: Juliette Baylon

Books on web development

Murach's HTML and CSS
Murach's JavaScript and jQuery
Murach's PHP and MySQL
Murach's ASP.NET Core MVC
Murach's Java Servlets and JSP

Books on programming languages

Murach's Python Programming
Murach's Java Programming
Murach's C#
Murach's C++ Programming

Books on data analysis

Murach's Python for Data Analysis

Books on SQL

Murach's MySQL
Murach's SQL Server for Developers
Murach's Oracle SQL and PL/SQL for Developers

For more on Murach books, please visit us at www.murach.com

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1
ISBN: 978-1-943872-86-2

Contents

Expanded contents

Chapter 3 How to use HTML to structure a web page

Chapter 4 How to use CSS to format the elements of a web page

Chapter 18 How to use JavaScript to enhance your web pages

Appendix A How to set up your computer for this book

Introduction

This 5th edition of our best-selling book integrates all the HTML and CSS skills that a web developer needs today with the proven instructional approach that made the first four editions so popular. And now, this edition simplifies, improves, and enhances the previous edition so it works better than ever.

In short, this is the right book if you're learning HTML and CSS for the first time. But this is also the right book if you're a web developer who wants to expand and update your skills. And after you used this book to learn, it becomes the best on-the-job reference you've ever used.

What this book does

- To get you started right, the first eight chapters present a subset of HTML and CSS that shows you how to develop web pages at a professional level. In chapter 3, you'll learn how to use HTML. In chapters 4 through 6, you'll learn how to use CSS to format the HTML. And in chapter 8, you'll learn how use Responsive Web Design to build web pages that look good and work right on every device: from mobile phone to tablet to desktop computer.
- When you finish the first 8 chapters, you will have the perspective and skills you need for developing professional web pages. Then, you can add to those skills by reading any of the chapters in the next three sections... and you don't have to read those sections or chapters in sequence. In other words, you can skip to any of the chapters in the last three sections after you finish section 1.
- The chapters in section 2 show you how to use two more approaches to Responsive Web Design. Chapter 9 presents Flexible Box Layout, chapter 10 presents Grid Layout, and you can use whichever approach is right for the types of web pages that you're creating.

- The chapters in section 3 let you learn new skills whenever you need them. If, for example, you want to learn how to use the data validation features for forms, you can skip to chapter 13. To learn how to add audio and video to your pages, skip to chapter 14. To learn how to use tables in your web pages, go back to chapter 12. To learn how to add custom fonts to your pages, go to chapter 11. And to learn how to use CSS transitions, transforms, animations, and filters, go to chapter 15.
- The chapters in section 4 present related skills that you can pick up whenever you're ready for them. In chapter 16, you can learn the basic principles for designing a website. In chapter 17, you can learn how to test and deploy a website and get it into the search engines. And in chapter 18, you can see how to use JavaScript and jQuery to enhance your web pages.

Why you'll learn faster and better with this book

Like all our books, this one has features that you won't find in competing books or online tutorials. That's why we believe you'll learn faster and better with our book than with any other. Here are a few of those features.

- From the first page to the last, this book shows you how to use HTML and CSS the modern, professional way, with HTML for the structure and content of each page and CSS for the formatting and page layout. That way, your web pages and your websites will be easier to create and maintain.
- Because HTML and CSS are integrated throughout the book, you won't learn these features out of context, which is the way they're often treated in competing materials. Instead, you'll learn exactly where these features fit into the overall context of website development.
- Because section 1 presents a complete subset of HTML and CSS, you are ready for productive work much sooner than you are when you use competing materials.
- If you page through this book, you'll see that all of the information is presented in "paired pages," with the essential syntax, guidelines, and examples for each topic on the right page and the perspective and extra explanation on the left page. This helps you learn faster by reading less... and this is the ideal format when you need to refresh your memory about how to do something.
- To show you how HTML and CSS work together, this book presents all the code for complete web pages that range from the simple to the complex. To see how that works, just page through chapters 4, 5, 6, and 8 to see the web pages that they present. As we see it, studying complete examples like these is the best way to master HTML and CSS because they show the relationships between the segments of code. And yet, most training materials limit themselves to snippets of code that don't show these relationships.
- Of course, this book also presents dozens of short examples. So it's easy to find an example that shows you how to do whatever you need to do as you

develop web pages. And our "paired pages" presentation method makes it much easier to find the example that you're looking for than it is with traditional presentations in which the code is embedded in the text.

What software you need

To develop web pages with HTML and CSS, you can use whichever text editor or IDE that you prefer. If you don't have a favorite, though, this book shows you how to use *Visual Studio Code* (*VS Code*). This text editor is widely popular, not only because it's free, but also because it provides many powerful features that will help you work faster and better. That's why Appendix A shows you how to install VS Code, and chapter 2 shows you how to use it.

Then, to test the web pages that you develop with this book, you should use at least two browsers. One of those should be Chrome, which is why Appendix A shows you how to install it. The other is the browser that comes with your computer: Edge for Windows or Safari for macOS.

How our downloadable files can help you learn

If you go to our website at www.murach.com, you can download all the files that you need for getting the most from this book. These files include:

- the HTML and CSS files for all of the applications and examples in this book
- the HTML and CSS files that you will use as the starting points for the exercises in this book
- the HTML and CSS files for the solutions to the exercises in the book

These files let you test, review, and copy code. In addition, if you have any problems with the exercises, the solutions are there to help you over the learning blocks, which is an essential part of the learning process. Here again, appendix A shows you how to download and install these files.

Support materials for trainers and instructors

If you're a corporate trainer or a college instructor who would like to use this book for a course, we offer supporting materials that include:

- a complete set of PowerPoint slides that you can use to review and reinforce the content of the book
- instructional objectives that describe the skills a student should have upon completion of each chapter
- test banks that measure mastery of those skills
- guided case studies that provide more exercises for your students (without the solutions)
- projects that your students both design and develop

If you're a college instructor and want to learn more about the instructor's materials, please go to our website at www.murachforinstructors.com. Or, if you're a trainer, go to www.murach.com and click on the Courseware for Trainers link. Another alternative is to call Kelly at 1-800-221-5528 or send an email to kelly@murach.com.

Please let us know how this book works for you

At long last, HTML and CSS have come of age, so there are no significant content additions in this edition of the book. Besides that, all modern browsers support all of the HTML and CSS features, so browser compatibility is no longer an issue that needs to be addressed.

That left us free to focus on simplifying and improving our figures and text so you can learn faster and better than ever from our book. And when you're through learning the HTML and CSS skills from this book, we want it to become the best on-the-job reference that you've ever used.

Now, we hope we've succeeded. We thank you for buying this book. We wish you all the best with your web development. And if you have any comments, we would appreciate hearing from you.

Zak Ruvalcaba
Author and content expert

Anne Boehm, Author
anne@murach.com

Section 1

The essential concepts and skills

The eight chapters in this section present the essential concepts and skills that you need for using HTML and CSS. These are the skills that you will use for almost every web page that you develop. And this is the minimum set of skills that every web developer should have.

When you complete this section, you'll be able to develop web pages at a professional level. Then, you can take your skills to the next level by reading the other sections and chapters in this book.

But please note that you don't have to read the chapters in the other sections in sequence. Instead, you can skip to any chapter that presents the skills that you want to learn next. In other words, the eight chapters in this section present the prerequisites for all of the other chapters in this book.

Chapter 1

Introduction to web development

This chapter introduces you to the concepts and terms that you need for working with HTML and CSS. When you finish this chapter, you'll have the background you need for learning how to build websites.

How web applications work

The *World Wide Web*, or web, consists of many components that work together to bring a web page to your desktop over the *Internet*. Before you start web pages of your own, you should have a basic understanding of how these components work together.

The components of a web application

The first diagram in figure 1-1 shows that web applications consist of *clients* and a *web server*. The clients are the computers, tablets, and mobile devices that use the web applications. They access the web pages through programs known as *web browsers*, such as Chrome, Edge, and Safari. The web server holds the files that make up a web application.

A *network* is a system that allows clients and servers to communicate. The Internet in turn is a large network that consists of many smaller networks. In a diagram like this, the "cloud" represents the network or Internet that connects the clients and servers.

In general, you don't need to know how the cloud works. But you should have a general idea of what's going on. That's why the second diagram in this figure gives you a conceptual view of the architecture of the Internet.

To start, networks can be categorized by size. A *local area network* (*LAN*) is a small network of computers that are near each other and can communicate with each other over short distances. Computers on a LAN are typically in the same building or in adjacent buildings. This type of network is often called an *intranet*, and it can be used to run web applications for use by employees only.

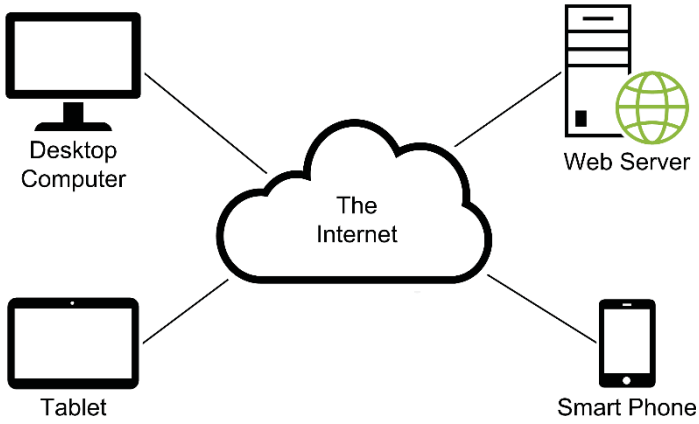
By contrast, a *wide area network* (*WAN*) consists of multiple LANs that have been connected together over long distances using *routers*. To pass information from one client to another, a router determines which network is closest to the destination and sends the information over that network. A WAN can be owned privately by one company or it can be shared by multiple companies.

An *Internet service provider* (*ISP*) is a company that owns a WAN that is connected to the Internet. An ISP leases access to its network to other companies that need to be connected to the Internet.

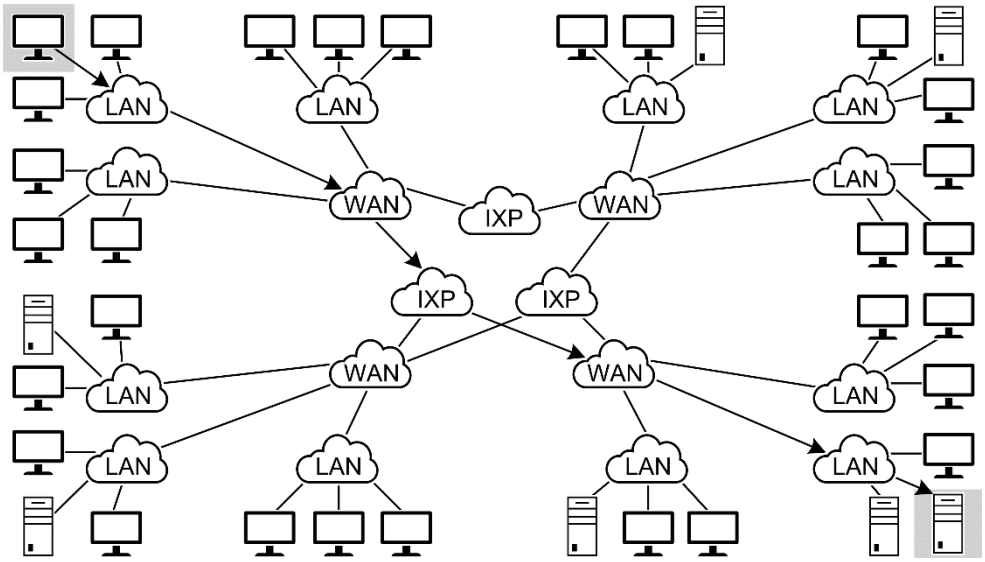
The Internet is a global network consisting of multiple WANs that have been connected together. ISPs connect their WANs at large routers called *Internet exchange points* (*IXP*). This allows anyone connected to the Internet to exchange information with anyone else.

If you study the diagram in this figure, you can get a better idea of how data is sent from the client in the top left to the server in the bottom right. First, the data leaves the client's LAN and enters the WAN owned by the client's ISP. Next, the data is routed through IXPs to the WAN owned by the server's ISP. Then, it enters the server's LAN and finally reaches the server. All of this can happen in less than 1/10th of a second.

The components of a web application



The architecture of the Internet



Description

- A web application consists of clients, a web server, and a network. The *clients* use programs known as *web browsers* to request web pages from the web server. The *web server* returns the pages that are requested to the browser.
- A *local area network* (LAN) directly connects computers that are near each other. This kind of network is often called an *intranet*.
- A *wide area network* (WAN) consists of two or more LANs that are connected by *routers*. The routers route information from one network to another.
- The *Internet* consists of many WANs that have been connected at *Internet exchange points* (IXPs). There are hundreds of IXPs located throughout the world.
- An *Internet service provider* (ISP) owns a WAN and leases access to its network. It connects its WAN to the rest of the Internet at one or more IXPs.

Figure 1-1 The components of a web application

How static web pages are processed

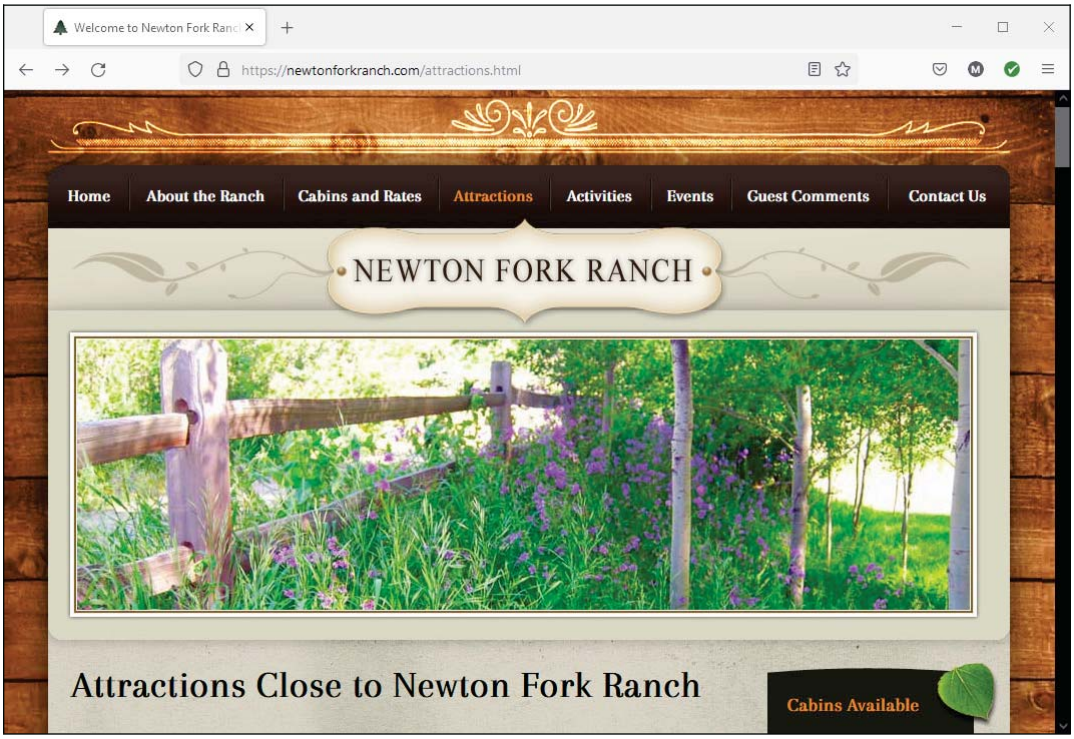
A *static web page* like the one at the top of figure 1-2 is a web page that is sent directly from the web server to the web browser when the browser requests it. This process begins when a client requests a web page using a web browser. To do that, the user can either type the address of the page into the browser's address bar or click a link in the current page that specifies the next page to load.

In either case, the web browser builds a request for the web page and sends it to the web server. This request, known as an *HTTP request*, is formatted using the *hypertext transfer protocol* (HTTP), which lets the web server know which file is being requested.

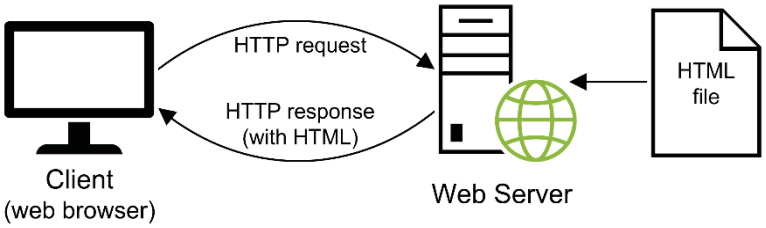
When the web server receives the HTTP request, it retrieves the requested file from the disk drive. This file contains the *HTML* (*HyperText Markup Language*) for the requested page. Then, the web server sends the file back to the browser as part of an *HTTP response*.

When the browser receives the HTTP response, it *renders* (translates) the HTML into a web page that is displayed in the browser. Then, the user can view the content. If the user requests another page, either by clicking a link or typing another web address into the browser's address bar, the process begins again.

A static web page at newtonforkranch.com/attractions.html



How a web server processes a static web page



Description

- *Hypertext Markup Language* (*HTML*) is used to define web pages.
- A *static web page* is an HTML document that's stored on the web server and doesn't change. The filenames for static web pages have .htm or .html extensions.
- When the user requests a static web page, the *web browser* sends an *HTTP request* to the web server that includes the name of the file that's being requested.
- When the web server receives the request, it retrieves the HTML for the web page and sends it back to the browser as part of an *HTTP response*.
- When the browser receives the HTTP response, it *renders* the HTML into a web page that is displayed in the browser.

Figure 1-2 How static web pages are processed

How dynamic web pages are processed

A *dynamic web page* like the one in figure 1-3 is a page that's created by a program called a *script* that runs on the web server each time it is requested. This script is executed by an *application server* based on the data that's sent with the HTTP request. In this example, the HTTP request identifies the book that's shown. Then, the script for the requested page retrieves the image and data for that book from a *database server*.

The diagram in this figure shows how a web server processes a dynamic web page. This process begins when the user requests a page in a web browser. To do that, the user can either type the URL of the page in the browser's address bar, click a link that specifies the dynamic page to load, or click a button that submits a form that contains the data that the dynamic page should process.

In each case, the web browser builds an HTTP request and sends it to the web server. This request includes whatever data the application needs for processing the request. If, for example, the user has entered data into a form, that data will be included in the HTTP request.

When the web server receives the HTTP request, the server examines the file extension of the requested web page to identify the application server that should process the request. The web server then forwards the request to the application server that processes that type of web page.

Next, the application server retrieves the appropriate script from the hard drive. It also loads any form data that the user submitted. Then, it executes the script. As the script executes, it generates the HTML for the web page. If necessary, the script will request data from a database server and use that data as part of the web page it is generating.

When the script is finished, the application server sends the dynamically generated HTML back to the web server. Then, the web server sends the HTML back to the browser in an HTTP response.

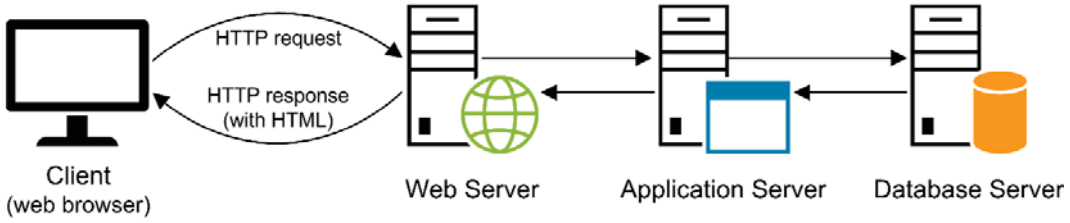
When the web browser receives the HTTP response, it renders the HTML and displays the web page. Note, however, that the web browser has no way to tell whether the HTML in the HTTP response was for a static page or a dynamic page. It just renders the HTML.

When the page is displayed, the user can view the content. Then, when the user requests another page, the process begins again. The process that begins with the user requesting a web page and ends with the server sending a response back to the client is called a *round trip*.

Dynamic web pages let you create interactive *web applications* that do all of the types of processing that you find on the Internet, including eCommerce applications. Although you won't learn how to develop dynamic web pages in this book, you will learn how to create the HTML forms that send user data to the web server. Then, after you master HTML, you can learn how to use server-side technologies like ASP.NET or PHP to create the dynamic pages that a website needs.

A dynamic web page at amazon.com

How a web server processes a dynamic web page



Description

- A *dynamic web page* is a web page that's generated by a program on the server that's called a *script*.
- When a web server receives a request for a dynamic web page, it looks up the extension of the requested file to find out which *application server* should process the request.
- When the application server receives a request, it runs the specified script. Often, this script uses the data that it gets from the web browser to get the appropriate data from a *database server*. This script can also store the data that it receives in the database.
- When the application server finishes processing the data, it generates the HTML for a web page and returns it to the web server. Then, the web server returns the HTML to the web browser as part of an HTTP response.

Figure 1-3 How dynamic web pages are processed

How JavaScript fits into web development

In contrast to the server-side processing that's done for dynamic web pages, *JavaScript* is a scripting language that provides for *client-side processing*. In the website in figure 1-4, for example, JavaScript is used to change the images that are shown without using server-side processing.

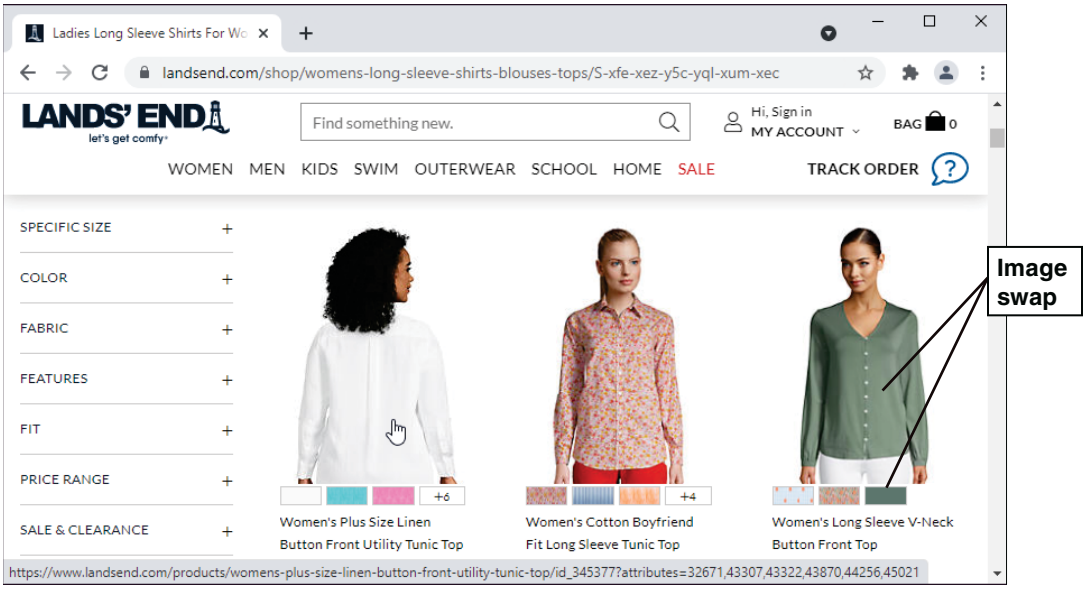
To make this work, all of the required images are loaded into the browser when the page is requested. Then, if the user clicks on one of the color swatches below a shirt, the shirt image is changed to the one with the right color. This is called an *image swap*. Similarly, if the user moves the mouse over a shirt, the image is replaced by the back view of the shirt. This is called an *image rollover*.

The diagram in this figure shows how JavaScript processing works. When a browser requests a web page, both the HTML and the related JavaScript are returned to the browser by the web server. Then, the JavaScript code is executed in the web browser by the browser's *JavaScript engine*. This takes some of the processing burden off the server and makes the application run faster. Often, JavaScript is used in conjunction with dynamic web pages, but it can also be used with static web pages.

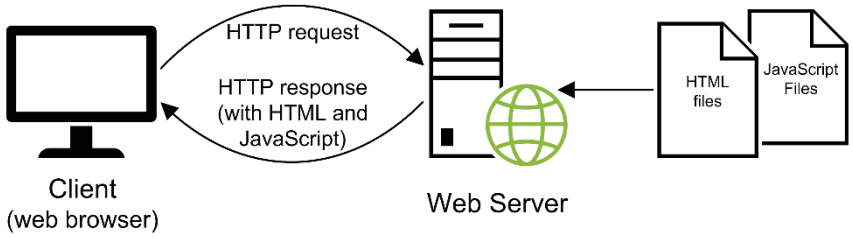
Besides image swaps and rollovers, there are many other uses for JavaScript. For instance, another common use is to validate the data that the user enters into an HTML form before it is sent to the server for processing. This saves unnecessary trips to the server. Other common uses of JavaScript are to provide for carousels and accordions.

In this book, you won't learn how to code JavaScript. However, you will learn how to use existing JavaScript routines in chapter 18 of this book. There, you'll learn how to use JavaScript and a JavaScript library known as jQuery to enhance your web pages with features like image swaps, image rollovers, and data validation.

A web page with image swaps and rollovers



How JavaScript fits into this architecture



Some common uses of JavaScript

- Data validation
- Image swaps and rollovers
- Carousels and accordions
- Slide shows

Description

- *JavaScript* is a *client-side scripting language* that is run by the *JavaScript engine* of a web browser.
- When the browser requests an HTML page that contains JavaScript or a link to a JavaScript file, both the HTML and the JavaScript are loaded into the browser.
- Because JavaScript runs on the client, not the server, it provides functions that don't require a trip back to the server. This can help an application run more efficiently.

Figure 1-4 How JavaScript fits into web development

An introduction to HTML and CSS

To develop a web page, you use HTML to define the content and structure of the page. Then, you use CSS to format that content. The topics that follow introduce you to HTML and CSS.

The HTML for a web page

HyperText Markup Language (*HTML*) is used to define the content and structure of a web page. In figure 1-5, for example, you can see the HTML for a simple web page, which can be called an *HTML document*. As you've already learned, this HTML is sent from a web server to a web browser running on a client. Then, the browser renders the HTML into a web page that's displayed in the browser.

Although you're going to learn how to code HTML in chapter 3, here's a quick introduction to how the HTML works. This document starts with a *DOCTYPE declaration* that is followed by *tags* that identify the *HTML elements* within the document. The *opening tag* for each element consists of the element name surrounded by angle brackets, as in <html>. And the *closing tag* consists of a left angle bracket, a forward slash, the element name, and the right angle bracket, as in </html>.

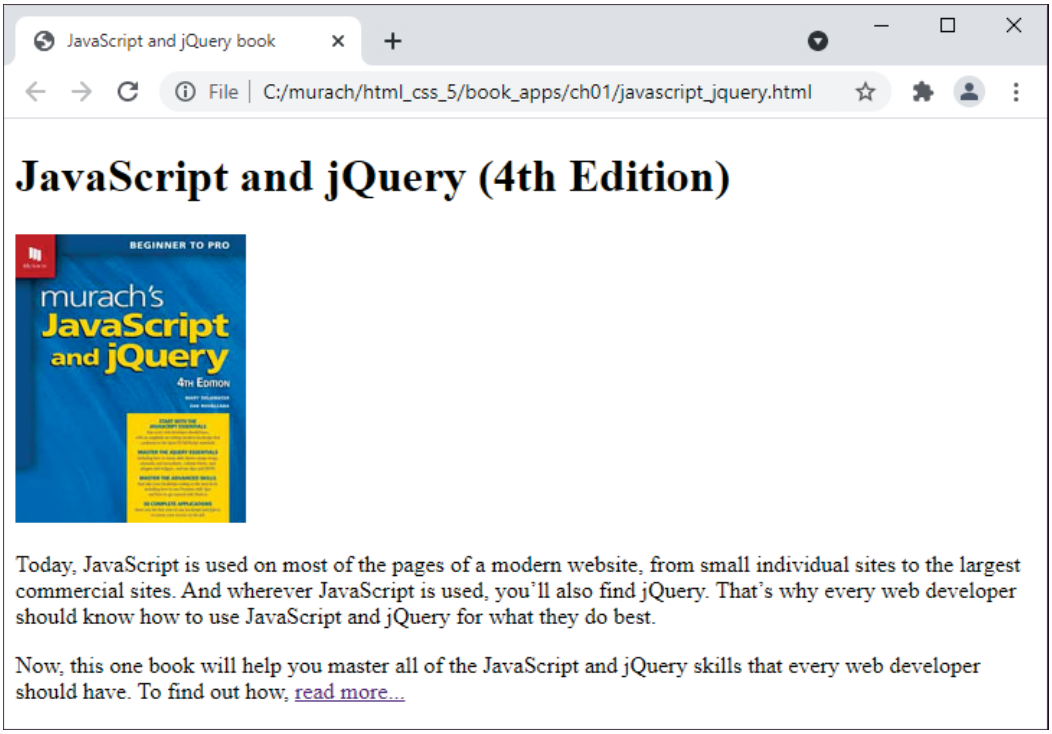
The basic structure of an HTML document consists of head and body elements that are coded within the html element. The head section contains elements that provide information about the document. The body section contains the elements that will be displayed in the web browser. For instance, the title element in the head section provides the title that's shown in the tab for the page in the web browser, while the h1 element in the body section provides the heading that's displayed in the browser window.

Many elements can be coded with *attributes* that identify the element and define the way the content in the element is displayed. These attributes are coded within the opening tag, and each attribute consists of an attribute name, an equals sign, and the attribute value. For instance, the <img> tag in this example has two attributes named src and alt. In this case, the src attribute provides the name of the image file that should be displayed, and the alt attribute provides the text that should be displayed if the image can't be found.

The code for an HTML file named javascript_jquery.html

```
<!doctype html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>JavaScript and jQuery book</title>
    </head>
    <body>
        <h1>JavaScript and jQuery (4th Edition)</h1>
        <img src="javascript_jquery.jpg" alt="JavaScript and jQuery Book">
        <p>Today, JavaScript is used on most of the pages of a modern
            website, from small individual sites to the largest commercial
            sites. And wherever JavaScript is used, you'll also find jQuery.
            That's why every web developer should know how to use JavaScript
            for what it does best and jQuery for what it does best.</p>
        <p>Now, this one book will help you master all of the JavaScript and
            jQuery skills that every web developer should have. To find out
            how, <a href="">read more...</a></p>
    </body>
</html>
```

The HTML displayed in a web browser



Description

- *HTML (HyperText Markup Language)* is used to define the structure and content of a web page.

Figure 1-5 The HTML for a web page

The CSS for a web page

To format the contents of a web page, you use *CSS* (C*ascading Style Sheets*). To do that, you can apply a CSS *style sheet* to an HTML document by a link element in the head section, as shown at the top of figure 1-6. Here, the href attribute of the tag says that the style sheet in the file named book.css should be applied to the HTML document.

After this link element, you can see the CSS that's in the book.css file. This is followed by a browser that shows how the web page is displayed after the style sheet has been applied to it. If you compare this to the browser in the previous figure, you can see that the page is now centered with a border around it, the font for the text has been changed, there's less spacing between paragraphs, and the text is displayed to the right of the book image. This gives you a quick idea of how much you can do with CSS.

Although you're going to learn how to code CSS in chapters 4, 5, and 6, here's a brief introduction to how the CSS works. First, this CSS file consists of four *style rules*. Each of these style rules consists of a *selector* and one or more *declarations* enclosed in braces { }. The selector identifies one or more HTML elements, and the declarations specify the formatting for the elements.

For instance, the first style rule applies to the body element. Its first declaration says that the font family for the content should be Arial, Helvetica, or the default sans-serif type, in that order of preference. Then, the second declaration says that the font-size should be 100% of the browser's default font size. These declarations set the base font and font size for the elements that are coded within the body.

The third declaration for the body sets its width to 560 pixels. Then, the fourth declaration sets the top and bottom margins to zero and the left and right margins to auto, which centers the page in the browser window. Finally, the fifth declaration sets the padding within the body to 1 em (a unit that you'll learn about in chapter 4), and the sixth declaration adds a solid, navy border to the body.

Similarly, the second style rule formats the h1 element in the HTML with a larger font size and the navy color. The third style rule formats the image by floating it to the left so the <p> elements are displayed to its right. And the fourth style rule changes the spacing between <p> elements.

This should give you an idea of how HTML and CSS work together. In short, the HTML defines the content and structure of the document, and the CSS defines the formatting of the content. This separates the content from the formatting, which makes it easier to create and maintain web pages.

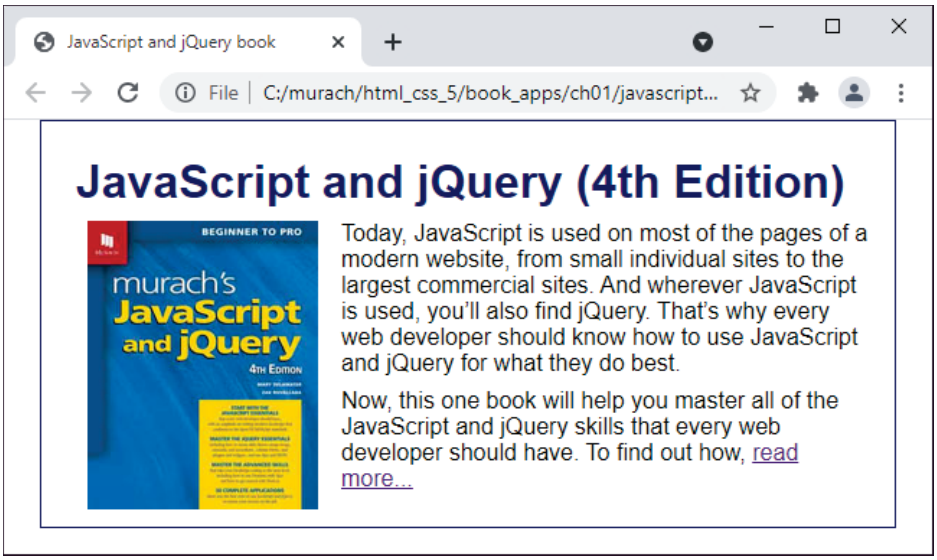
The element in the head section of the HTML file that links it to the CSS file

```
<link rel="stylesheet" href="book.css">
```

The code for the CSS file named book.css

```
body {
    font-family: Arial, Helvetica, sans-serif;
    font-size: 100%;
    width: 560px;
    margin: 0 auto;
    padding: 1em;
    border: 1px solid navy;
}
h1 {
    margin: 0;
    padding: .25em;
    font-size: 200%;
    color: navy;
}
img  {
    float: left;
    margin: 0 1em 1em 1em;
}
p {
    margin: 0;
    padding-bottom: .5em;
}
```

The web page displayed in a web browser



Description

- *Cascading Style Sheets* (*CSS*) are used to control how web pages are displayed by specifying the fonts, colors, borders, spacing, and layout of the pages.

Figure 1-6 The CSS for a web page

A short history of the HTML and CSS standards

In case you're interested, figure 1-7 presents a short history of the HTML and CSS standards. As you can see, HTML standards have been around since 1995, but they didn't get stabilized until version 5, which was adopted in 2014. This version is commonly referred to as HTML5.

Similarly, CSS standards have been around since 1996, but they didn't get stabilized until version 3 and that version didn't get widespread use until 2010 or later. This version is commonly referred to as CSS3.

In recent years, new features were added to HTML5 and CSS3, but without new release numbers. As a result, there's no longer any reason to include the version numbers when referring to them. So, from this point on, this book refers to HTML5 and CSS3 code as just HTML and CSS, unless there is a reason to refer to the version number.

This figure also presents two websites that you ought to become familiar with. The first is for the *World Wide Web Consortium*, which is commonly referred to as *W3C*. Until May of 2019, this was the group that developed the standards, and this site is a primary source for HTML and CSS information.

The second website is for the *Web Hypertext Application Technology Working Group* (*WHATWG*). This is a community of people interested in evolving HTML and related technologies, and this site is another primary source for HTML and CSS information. In May 2019, this group took over the development of the HTML standards, although the W3C continues to participate in the development process.

Highlights in the development of the HTML standards

Version	Description
HTML 1.0 to 4.01	HTML 2.0 was the first specification adopted as a standard by the W3C in November 1995. HTML 4.0 and 4.01 added new features and deprecated older features.
XHTML 1.0 to 1.1	XHTML 1.0 was adopted in January 2000 and reformulated HTML 4 using the syntax of XML. With XHTML 1.1, the control of the presentation of content was now done through CSS.
HTML 5 to 5.2	HTML 5 was adopted in October 2014 and replaced the current versions of both HTML and XHTML. HTML 5.1 and 5.2 were minor revisions of these standards.
HTML Living Standard	The WHATWG started developing these standards as a split from the W3C in July of 2012. In May 2019, the W3C ceded authority over the HTML standards to WHATWG.

Highlights in the development of the CSS standards

Version	Description
1.0	Adopted in December 1996.
2.0	Adopted in May 1998.
2.1	First released as a candidate standard in February 2004, it returned to working draft status in June 2005. It became a candidate standard again in July 2007.
3.0	A modularized version of CSS with the earliest drafts in June 1999. Some modules build on existing features of CSS 2.1, and others provide entirely new features. Each module is accepted as a standard independently.

Two websites that you should become familiar with

- The *World Wide Web Consortium* (*W3C*) is an international community in which member organizations, a full-time staff, and the public work together to help develop Web standards. Its website address is https://w3.org.
- The *Web Hypertext Application Technology Working Group* (*WHATWG*) is a community of people interested in evolving HTML and related technologies, and it currently maintains the HTML standards. Its website address is: https://html.spec.whatwg.org.

Description

- The W3C ceded authority over the HTML standards to the WHATWG in May 2019 after determining that having two standards was harmful. However, the W3C still participates in the development process.
- Unlike the W3C standards, the WHATWG Living Standard is continually evolving. Because of that, it doesn't use version numbers. Today, all modern browsers support the Living Standard.
- The last version numbers for HTML and CSS were HTML5 and CSS3, but you don't need to specify the version numbers any more. As a result, they aren't used in the rest of this book.

Figure 1-7 A short history of the HTML and CSS standards

Tools for web development

To create and edit the HTML and CSS files for a website, you need either a text editor or an IDE for web development. To deploy a website on the Internet, you also need an FTP client that lets you upload files from your computer or network server to the web server. You'll learn about these tools next.

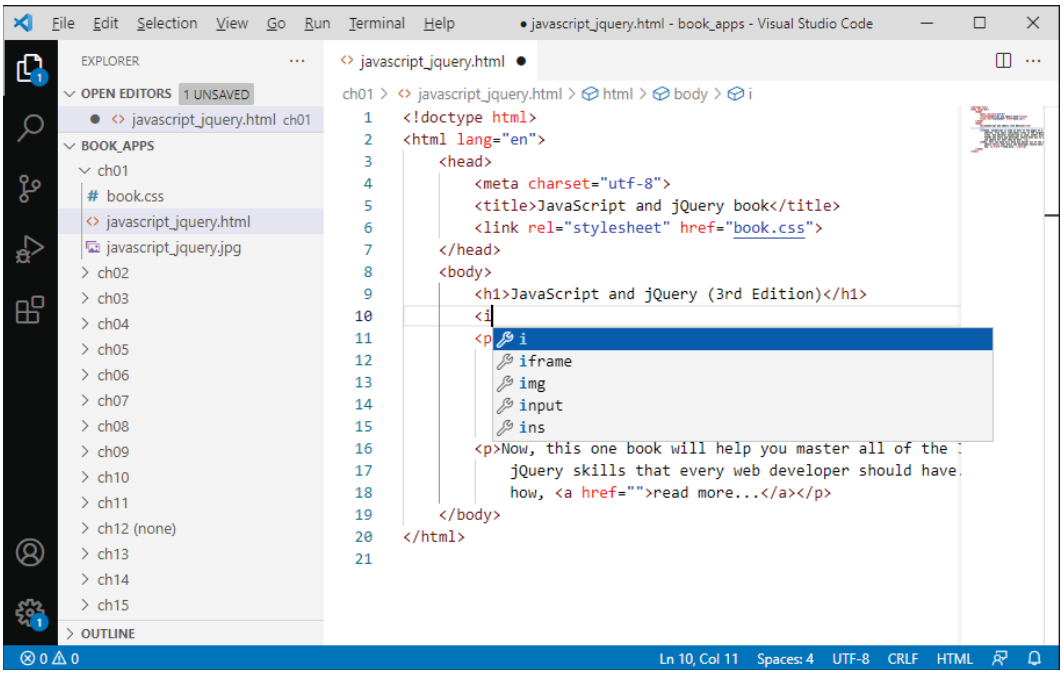
Text editors and IDEs

A *text editor* lets you enter and edit HTML and CSS, and figure 1-8 lists four of them. Of these, we recommend that you use *Visual Studio Code* (or just *VS Code*). It is a free editor that runs on both Windows and macOS systems; it has many excellent features; and it will help you work faster and better.

In this figure, for example, you can see how VS Code provides an auto-completion list that lets you select an item after you enter the first character or two. To help you get started with VS Code, appendix A shows how to install it, and chapter 2 presents a short tutorial on how to use it.

The alternative to a text editor is an *Integrated Development Environment* (*IDE*) for web development. As this figure shows, two of the most popular IDEs for web development are Adobe Dreamweaver and AWS (Amazon Web Services) Cloud9. As you would expect, an IDE provides all the features of a text editor plus other features like a built-in FTP client, which you'll learn about in the next figure.

VSCode with the auto-completion feature in progress



Four of the text editors that you can use for web development

Editor	Runs on
VS Code	Window, macOS, and Linuxs
Notepad++	Windows
TextMate	macOS
Codepen	The Web

Two of the IDEs for web development

IDE	Runs on
Adobe Dreamweaver	Windows and macOS
AWS Cloud9	The Web

Description

- A *text editor* lets you enter and edit the HTML and CSS files for a web application. Some common features of a text editor are syntax highlighting and auto-completion.
- An *Integrated Development Environment* (*IDE*) goes beyond text editing to provide other features for the development of websites, like a built-in FTP program (see the next figure).

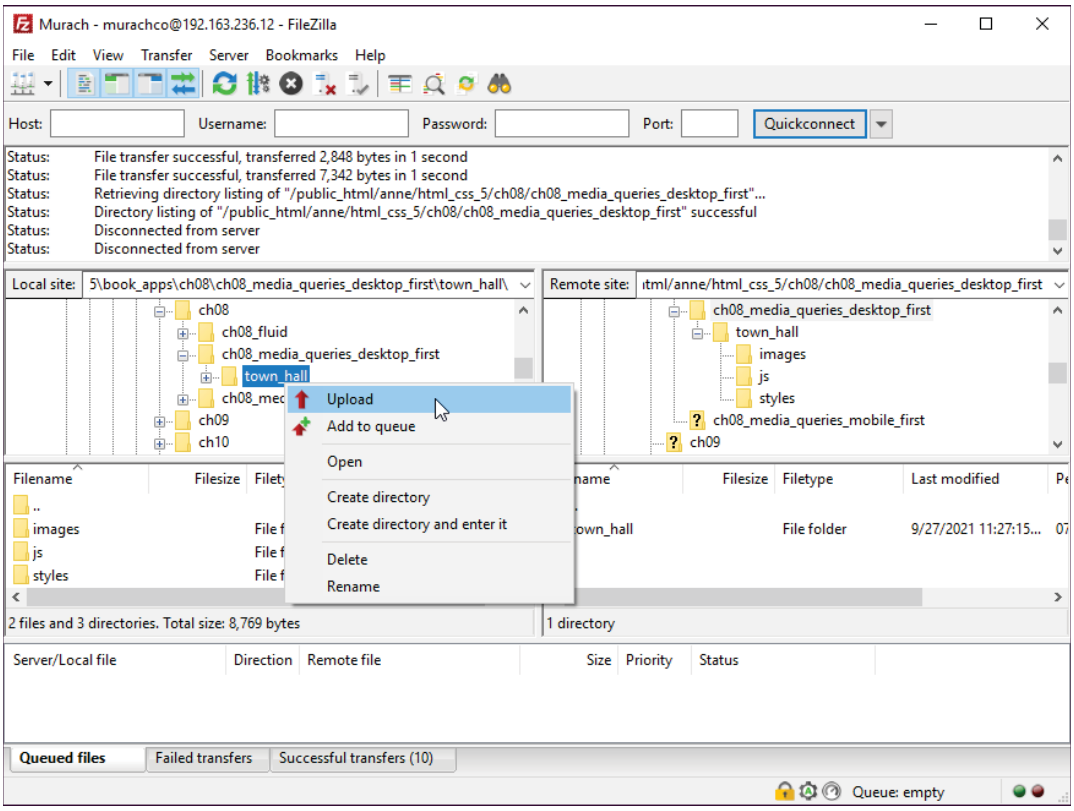
Figure 1-8 Text editors and IDEs for web development

FTP clients

If you want to *deploy* (or *publish*) your website on the Internet, you need to transfer the folders and files for your website from your computer or network to a web server with Internet access. One way to do that is to use an *FTP client* that uses *File Transfer Protocol* (or *FTP*) to transfer the folders and files. Figure 1-9, for example, shows an FTP client named FileZilla as it uploads files to a web server.

Note, however, that FileZilla is just one of many FTP clients that you can use for uploading files. Remember too that IDEs like Dreamweaver have built-in FTP clients.

FileZilla as it is used to upload files to the web server



Some free FTP clients

Program	Runs on
FileZilla	Windows, macOS, and Linux
FireFTP	Windows, macOS, and Linux
Classic FTP	Windows and macOS

Description

- To *deploy* (or *publish*) a website on the Internet, you can use *File Transfer Protocol* (or *FTP*) to transfer the folders and files for the website from your computer or local network to a web server on the Internet. To do that, you can use an *FTP client*.
- If you're using a text editor, you typically have to use a separate FTP client or add a plugin FTP client to your editor. In contrast, most IDEs have built-in FTP clients.

Figure 1-9 FTP clients for uploading files to the web server

How to view a web page and its source code

Next, you'll learn how to view a web page in a web browser and how to view the source code for a web page that's displayed in the browser. These are valuable skills that you can use when you test your own web pages or study the web pages on other sites.

How to view a web page

Figure 1-10 shows you how to view a web page on the Internet by entering a a *uniform resource locator* (*URL*) into the address bar of your browser. As the diagram at the start of this figure shows, the URL for an Internet page can include four components. In most cases, the *protocol* is HTTP. But if you omit the protocol, the browser uses HTTP as the default.

The second component is the *domain name* that identifies the web server that the HTTP request will be sent to. The web browser uses this name to look up the address of the web server for the domain. Although you can't omit the domain name, you can usually omit the www and the dot that precede it.

The third component is the *path* that lists the folders on the server that contain the file. Forward slashes are used to separate the names in the path and to represent the server's top-level folder at the start of the path. In this example, the path is:

```
/courseware-for-trainers/what-our-courseware-includes
```

The last component is the name of the file. But if you omit the filename, as in this example, the web server will search for a default document in the path. Depending on the web server, this file will be named index.html, default.htm, or some variation of the two.

If you want to view an HTML page that's on your own computer or a local network, you can use one of the two techniques that are presented next. First, your text editor or IDE should provide a way that makes it easy to view a page that you're working on. Second, you can find the file in your file explorer, and then double-click on it to open it in your default browser. Or, you can right-click on it and select the browser that you want to open it with.

At the bottom of this figure, you can see our naming recommendations for your folders and files. In general, your folder and file names should only contain lowercase letters, numbers, underscores or hyphens, and the period. In the examples in this book, you'll see the author's preference, which is to use underscores instead of hyphens to separate the words in a name. But many developers use hyphens instead of underscores.

The other recommendation is to create names that clearly indicate the contents of your folders and web pages. That can improve search engine optimization (SEO), which you'll learn more about in a moment.

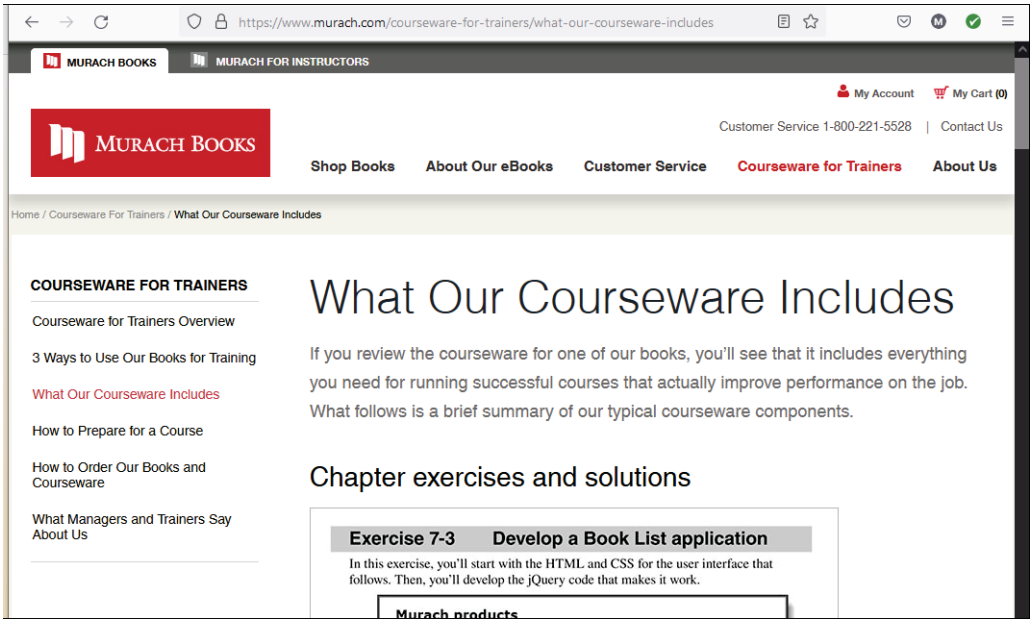
The components of an HTTP URL

```
protocol://domain-name/path/filename
```

A URL with an omitted filename

http://www.murach.com/courseware-for-trainers/what-our-courseware-includes

The web page at that URL



What happens if you omit parts of a URL

- If you omit the protocol, the default of http will be used.
- If you omit the filename, the default document name for the web server will be used. This is typically index.html, default.htm, or some variation.

How to access a web page on the Internet

- Enter the URL of a web page into the browser's address bar.

How to access a web page on your own server or computer

- Use the features of your text editor or IDE.
- Find the file in your file explorer. Then, double-click on it to open it in your default browser. Or, right-click on it and use the Open With command to select the browser.

Naming recommendations for your own folders and files

- Create names for folders and files that consist of lowercase letters, numbers, underscores or hyphens, and the period.
- Use filenames that clearly indicate what a page contains. This is good for search engine optimization (see figure 1-15).

Figure 1-10 How to view a web page

How to view the source code for a web page

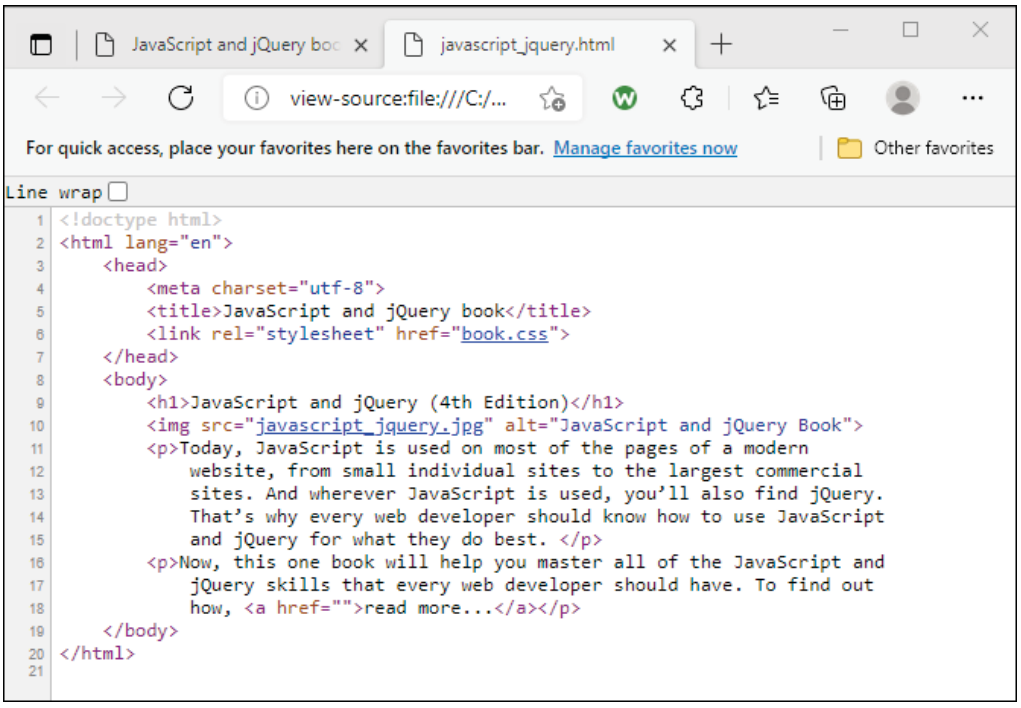
When a web page is displayed by a browser, you can use the first technique in figure 1-11 to view the HTML for the page. You just right-click on the page and select View Page Source. Then, if you want to view the CSS for the page, you can click on the CSS file or files that the HTML page is linked to. Or, if the CSS is coded in the head element of the HTML file, you can see it there.

In this example, the source code for the web page in figure 1-6 is displayed in a new tab that was opened for it. At this point, you can click on the book.css link to open the CSS file in another new tab.

In some cases, viewing the source code can be useful when you're testing and debugging a web page. At the least, it lets you confirm that the browser is running the HTML and CSS that you've coded in your text editor or IDE. Sometimes, this helps you discover problems that you wouldn't have imagined. This is especially true when you're debugging dynamic web pages that were generated by a script.

You may also want to view the source code for the pages on other Internet sites so you can study it. That can be a good way to learn how other sites work. Although some sites use various techniques to hide their code, a lot of the code for Internet sites is available. But beware, the code for all but the simplest of sites can get complicated in a hurry. That's partly because the code is generated by a content management system.

The source code for the web page in figure 1-6




```
<!doctype html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>JavaScript and jQuery book</title>
        <link rel="stylesheet" href="book.css">
    </head>
    <body>
        <h1>JavaScript and jQuery (4th Edition)</h1>
        <img src="javascript_jquery.jpg" alt="JavaScript and jQuery Book">
        <p>Today, JavaScript is used on most of the pages of a modern
            website, from small individual sites to the largest commercial
            sites. And wherever JavaScript is used, you'll also find jQuery.
            That's why every web developer should know how to use JavaScript
            and jQuery for what they do best. </p>
        <p>Now, this one book will help you master all of the JavaScript and
            jQuery skills that every web developer should have. To find out
            how, <a href="">read more...</a></p>
    </body>
</html>
```

How to view the HTML for a web page

- Right-click the page and select View Page Source from the popup menu.

How to view the CSS if it's in an external CSS file

- Click on the link that refers to the CSS file.

How to view the CSS if it's in the HTML file

- You'll find it in the head element of the file.

Description

- When you're debugging your own web pages, it is sometimes useful to view the HTML and CSS for a page that's being displayed in a browser. At the least, this can confirm that the browser is displaying the page that you think it's displaying.
- This also provides a way to see how other websites are coded. In general, though, the code for large, commercial websites is extremely complicated. That's often because it's generated by a content management system like Joomla.

Figure 1-11 How to view the source code for a web page

Four critical web development issues

Whenever you develop a web application, you should be aware of the issues that are presented in the next four figures. Then, as you progress through this book, you will learn how to provide for each of them.

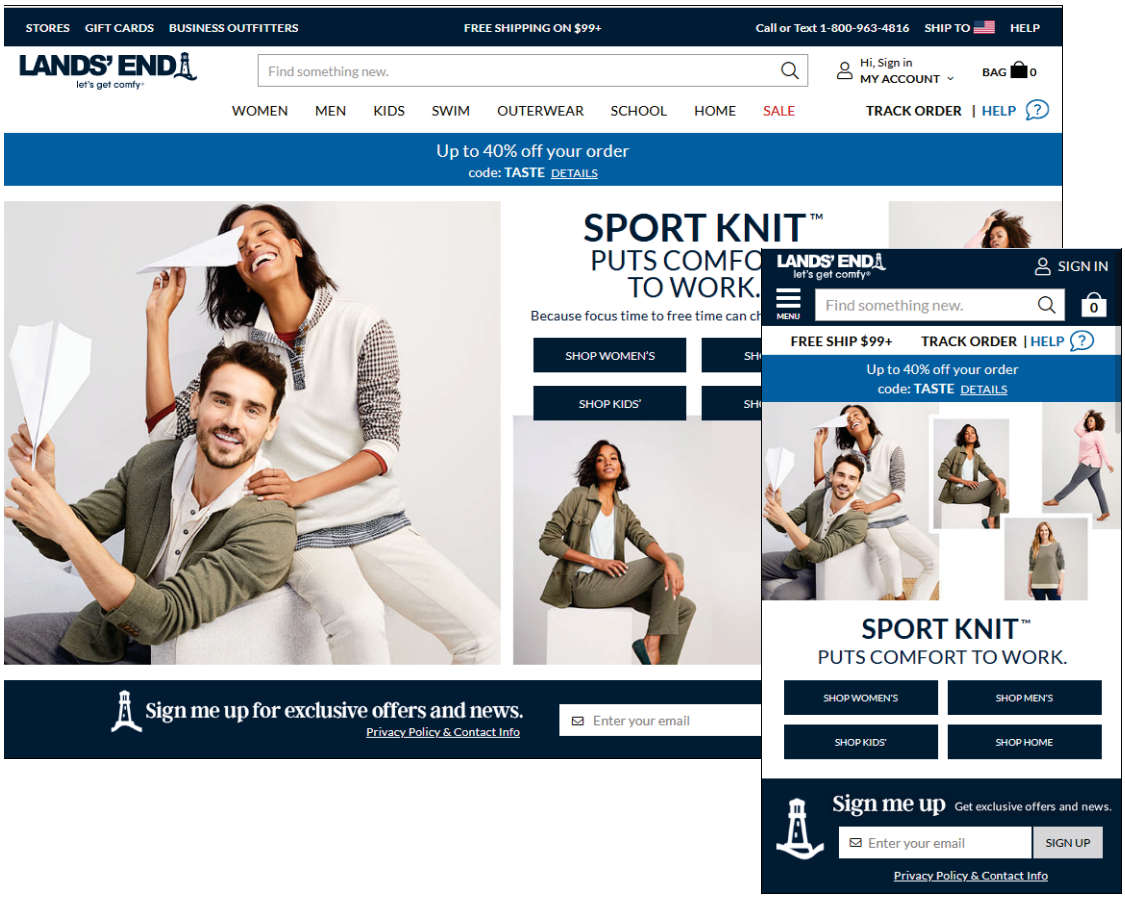
Responsive Web Design

Responsive Web Design, or *RWD*, just means that a website should work on all devices that access it: from desktop computers to tablets to mobile phones in landscape mode, to mobile phones in portrait mode. The statistics in figure 1-12 help explain why that's important, and you can find many other statistics along the same lines. But the need for RWD is so obvious that it shouldn't need statistical support.

This figure illustrates how a site that uses RWD adapts to the size and orientation of the screen. Here, you can see the home page of the Lands' End website in a desktop browser and also on a mobile phone in portrait mode. This shows that the look-and-feel of the page remains the same in both screen sizes. And that's the beauty of Responsive Web Design!

Because RWD is such an important subject, this book has three chapters that show you how to implement it. In chapter 8, you'll learn the essential concepts and skills. Then, chapter 9 shows you how to use Flexible Box Layout to implement RWD, and chapter 10 shows you how to use Grid Layout to implement it.

The Lands' End home page on a desktop computer and a mobile phone



Four of the many statistics that prove the need for Responsive Web Design

- In a global survey of 87,000 users, 95% used mobile phones to access the Internet, 93% used desktops, and 73% used tablets.
- 57% of all web traffic comes from mobile devices.
- 50% of all web shoppers in the United States buy from mobile devices.
- 40% of the mobile users will go to a different site if the first one isn't mobile friendly.

Description

- *Responsive Web Design* means that a website should adapt to the screen size of the device that's accessing it, whether it's a desktop computer, a tablet, or a mobile phone.

Figure 1-12 Responsive Web Design

Cross-browser compatibility

If you want your website to be used by as many visitors as possible, you need to make sure that your web pages are compatible with all the browsers that people may use to access your website. That's known as *cross-browser compatibility*.

Although this was a big issue just a few years back, modern browsers support almost all of the features that are provided by the latest versions of HTML and CSS. This is indicated by the table in figure 1-13 that rates the HTML compatibility of the five browsers that account for almost all of the desktop Internet activity in the US and throughout the world. To get an updated version of this information, you can go to the website at www.html5test.com.

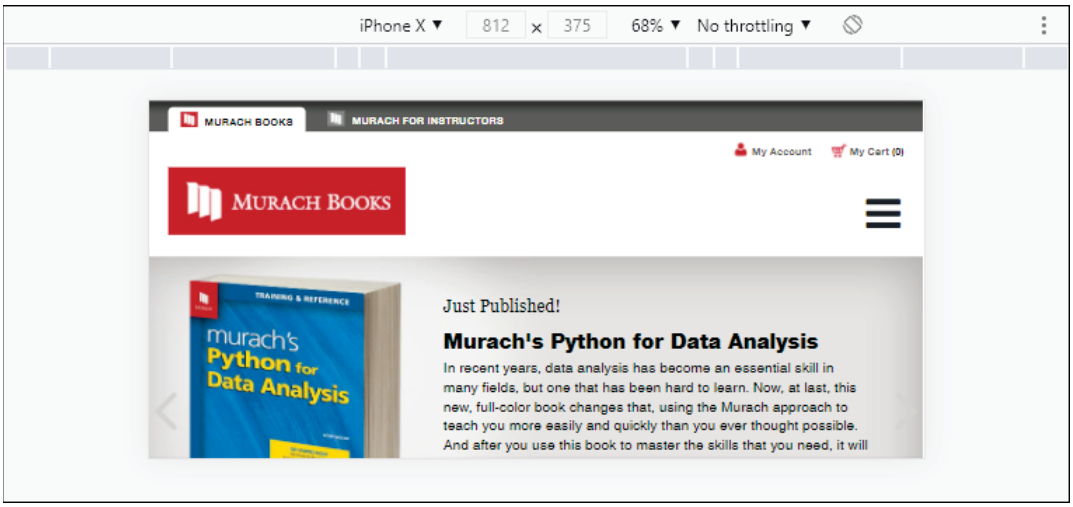
In the past, Internet Explorer gave web developers the most problems because it was the least standard and didn't provide for automatic updates. In contrast, the five desktop browsers in this figure provide for automatic updates so you don't have to wonder whether they include the latest HTML and CSS features. Besides that, Internet Explorer has such a trivial share of the market today that you no longer have to support it.

What's new is that now you have to worry about tablet and mobile browsers too. So, how do you provide cross-browser compatibility in today's world? First, you should of course use the current versions of HTML and CSS as you develop your web pages. If you do that, with the exception of just a few features, everything that you learn in this book will run on all web browsers. And for those few features that aren't implemented yet by one or more of the browsers, this book provides an appropriate workaround.

Second, you should test your pages on desktop, tablet, and mobile browsers. The good news is that the five desktop browsers in this figure provide Developer Tools that make that easy. For instance, the example at the top of this figure shows how you can use Chrome's Developer Tools to display a working version of a web page in mobile landscape format. You'll learn more about this in chapter 8.

Eventually, you'll want to test your web pages on the actual browsers too. But since Chrome, Edge, and Opera are all Chromium-based, you only need to test on one of them. Then, you can test on Firefox and Safari just to be sure that they don't present any problems. And for good measure, you can test your web pages on tablets and mobile phones...if for no other reason than to see how well they work on those devices.

Chrome's Developer Tools with a page in mobile landscape format



The desktop browsers and their HTML ratings (perfect score is 555)

Browser	Release	HTML5 Test Rating
Google Chrome	91	528
Opera	63	518
Edge	91	492
Mozilla Firefox	90	491
Apple Safari	11	471

The website for these ratings

http://www.html5test.com

Coding guidelines

- Use the latest versions of HTML and CSS.
- Use the workarounds that are presented in this book for any features that aren't supported by all current browsers.

Testing guidelines

- Test your web pages on desktop browsers as well as tablet and mobile phone browsers.
- You can use Chrome's Developer Tools for the initial testing of your tablet and mobile layouts (see figure 8-2 in chapter 8).

Description

- *Cross-browser compatibility* means that your web pages will work on any browser that accesses your website, including tablet and mobile phone browsers.
- At one time, this was a major development issue. But today, with just a few exceptions, all of the current browsers support all of the features that are presented in this book.

Figure 1-13 Cross-browser compatibility

Web accessibility

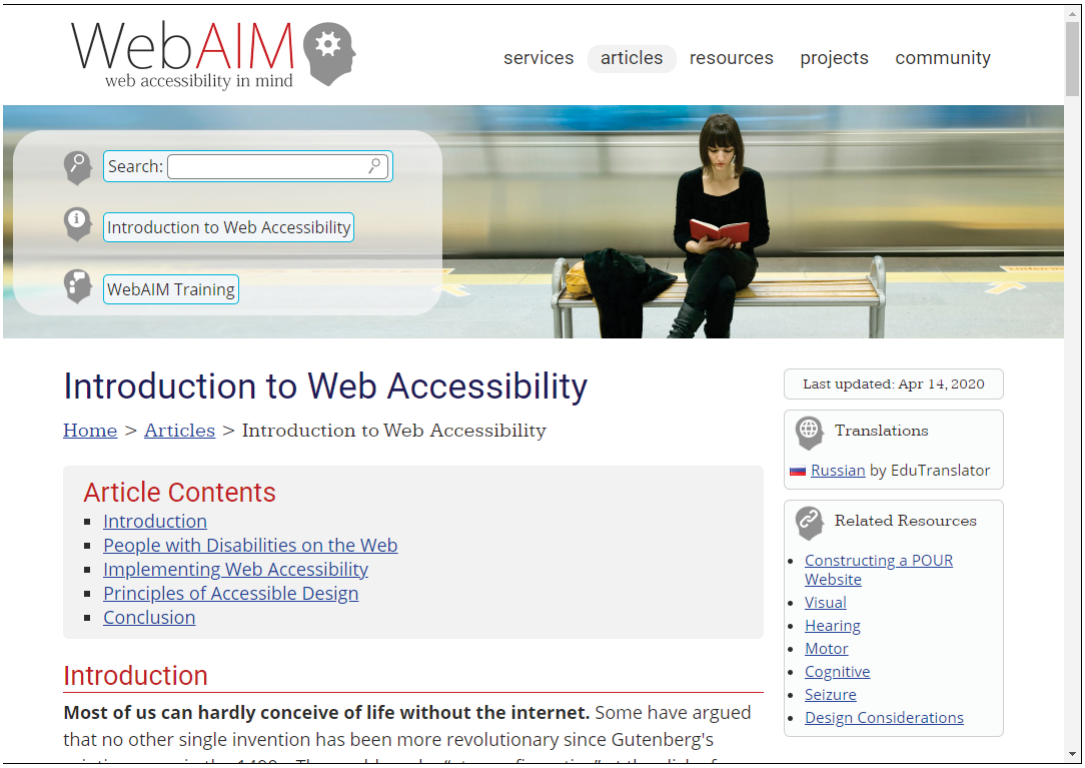
Web accessibility (or just *accessibility*) is described in figure 1-14. It refers to the qualities that make a website accessible to as many users as possible, especially disabled users.

For instance, visually-impaired users may not be able to read text that's in images, so you need to provide other alternatives for them. Similarly, users with motor disabilities may not be able to use the mouse, so you need to make sure that all of the content and features of your website can be accessed through the keyboard.

To a large extent, this means that you should develop your websites so the content is still usable if images, CSS, and JavaScript are disabled. A side benefit of doing that is that your site will also be more accessible to search engines, which rely primarily on the text portions of your pages.

In this book, you will be given guidelines for providing accessibility as you learn the related HTML. And to learn more about accessibility, we recommend that you go to the sites that are identified in this figure. As the example in this figure shows, the WebAim website is a good place to start because it presents an excellent introduction to accessibility.

The WebAIM website



Accessibility laws that you should be aware of

- The Americans with Disabilities Act (ADA).
- Sections 504 and 508 of the federal Rehabilitation Act.
- Section 255 of the Telecommunications Act of 1996.

Types of disabilities

- Visual
- Hearing
- Motor
- Cognitive

Information sources for accessibility

- The WebAIM website: http://www.webaim.org.
- The World Wide Web Consortium (W3C): http://www.w3.org/TR/WCAG.
- W3C provides an accessibility specification called WAI-ARIA that shows how to make applications more accessible: http://www.w3.org/TR/wai-aria.

Description

- *Web accessibility* refers to the qualities that make a website accessible to users, especially disabled users.
- As you go through this book, you'll be given guidelines for coding the elements and attributes that provide web accessibility.

Figure 1-14 Web accessibility

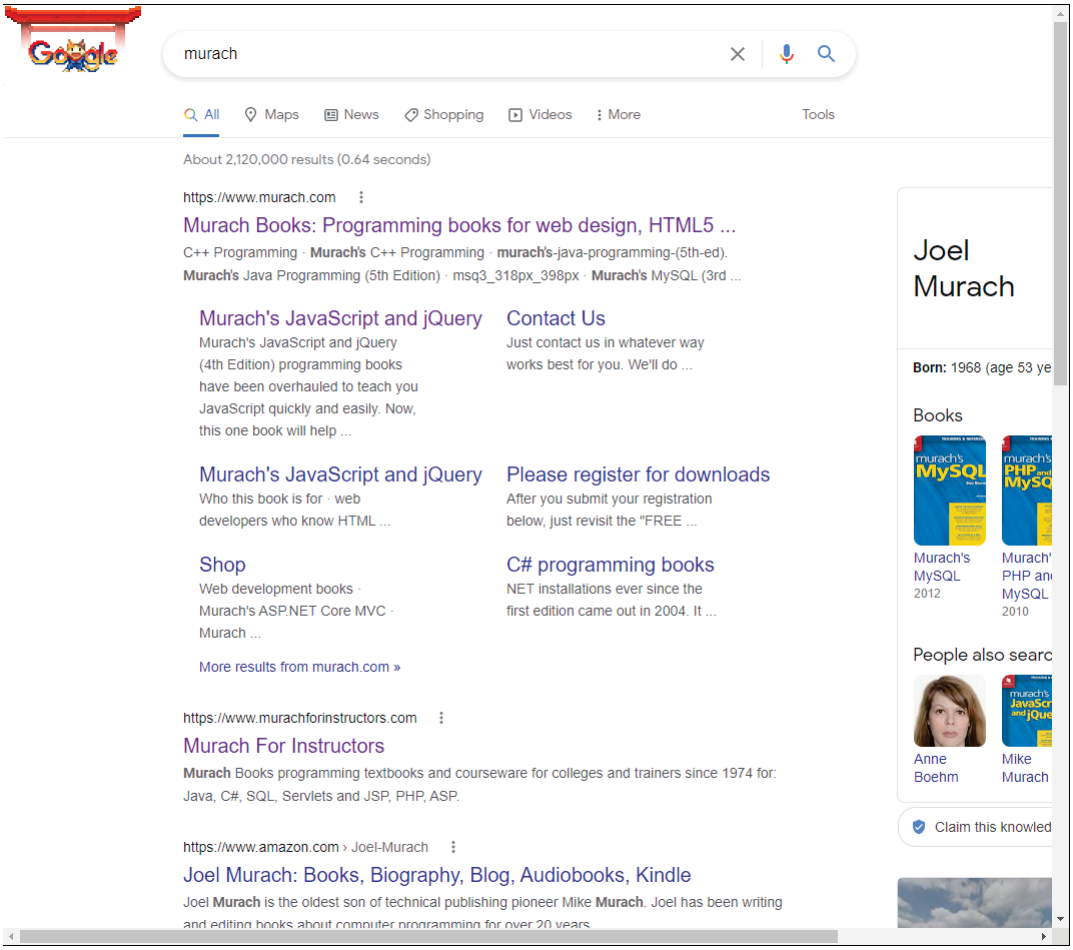
Search engine optimization

Search engine optimization, or *SEO*, is described in figure 1-15. It refers to the goal of optimizing your website so your pages rank higher in search engines like Google and Bing.

The example in this figure shows how important SEO can be. Here, the search term is "murach" so we would hope that our website would be the first item listed...and it is. But note that the third listing is for the Amazon website, which also sells our books. Now, imagine how we would feel if we couldn't find our site on the first page of listings.

Because the algorithms that are used by search engines are changed frequently, optimizing your website and web pages requires a continual effort. That's why large development groups have SEO specialists on their staffs. To get you started right, though, this book presents the coding practices for improved SEO that don't change.

The results of a Google search for “murach”



The most popular search engines

- Google
- Bing

Description

- *Search engine optimization* (*SEO*) refers to the goal of optimizing your website so its pages will rank high in the search engines that are used to access them.
- Although the search algorithms that are used by the search engines are changed frequently, this book presents the common coding techniques that will help your pages do better in the search engines.

Figure 1-15 Search engine optimization

Perspective

Now that you know the concepts and terms that you need for developing websites with HTML and CSS, you're ready to learn how to develop a web page. So in the next chapter, you'll learn how to create, test, and validate a web page. After that, you'll be ready to learn all the details of HTML and CSS.

Terms

client
web browser
web server
local area network (LAN)
intranet
wide area network (WAN)
Internet
HTML (HyperText Markup Language)
static web page
HTTP request
HTTP response
HTTP (HyperText Transfer Protocol)
render a web page
dynamic web page
script
application server
database server
round trip
JavaScript
client-side processing
JavaScript engine
HyperText Markup Language (HTML)
HTML document
HTML element
CSS (Cascading Style Sheets)
style sheet
style rule
text editor
IDE (Integrated Development Environment)
deploy (publish)
FTP (File Transfer Protocol)
FTP client
URL (Uniform Resource Locator)
Responsive Web Design (RWD)
cross-browser compatibility
web accessibility
search engine optimization (SEO)

Summary

- A web application consists of clients, a web server, and a network. *Clients* use *web browsers* to request web pages from the web server. The *web server* returns the requested pages.
- A *local area network* (*LAN*) connects computers that are near to each other. This is often called an *intranet*. By contrast, a *wide area network* (*WAN*) uses *routers* to connect two or more LANs. The *Internet* consists of many WANs.
- To request a web page, the web browser sends an *HTTP request* to the web server. Then, the web server retrieves the HTML for the requested page and sends it back to the browser in an *HTTP response*. Last, the browser *renders* the HTML into a web page.
- A *static web page* uses the same HTML each time it is accessed. In contrast, the HTML for a *dynamic web page* is generated by a server-side *script*, so its HTML can change from one request to another.

- *JavaScript* is a *scripting language* that is run by the *JavaScript engine* of a web browser. It provides for *client-side processing*.
- *HTML* (*HyperText Markup Language*) is the language that defines the structure and content of a web page. *CSS* (*Cascading Style Sheets*) is used to control how the web pages are formatted.
- To develop web pages, you can use a *text editor* like Visual Studio Code or an *Integrated Development Environment* (*IDE*) like Dreamweaver.
- To *deploy* (or *publish*) a website on the Internet, you need to transfer the folders and files for your site from your computer to a web server with Internet access. To do that, you use an *FTP client* that uses *File Transfer Protocol* (*FTP*).
- To view a web page on the Internet, you can enter the *URL* (*Uniform Resource Locator*) into a browser's address bar.
- To view a web page that's on your own computer or server in a browser, you can use the features of your text editor or IDE. Or, you can find the file in your file explorer and double-click or right-click on it.
- To view the HTML for a web page, right-click on the page and select View Source or View Page Source. Then, to view the CSS for a page, you can click on its link in the source code.
- Four critical web development issues are *Responsive Web Design* (*RWD*), *cross-browser compatibility*, *web accessibility*, and *search engine optimization* (*SEO*).

Before you do the exercises for this book...

Before you do the exercises for this book, you should download and install the Chrome browser. You should also download and install the applications, examples, and exercises for this book. The procedures for doing both are in appendix A.

Exercise 1-1 View the example in this chapter

In this exercise, you'll visit the book page in figures 1-5 and 1-6.

1. Use File Explorer or Finder to locate this HTML file:
   `\html_css_5\book_apps\ch01\javascript_jquery.html`
   Then, double-click on the file to open it in your default browser.
2. If Chrome isn't your default browser, use your file explorer to locate the file in step 1 again. This time, right-click on it and use the Open With command to open the file in Chrome.
3. In Chrome, right-click on the page and choose View Page Source to display the source code for this page in a new tab. Then, click on book.css in the link element in the HTML code to display the CSS file for this page in a new tab.

Exercise 1-2 View book apps and examples

This exercise will introduce you to the types of examples and applications that you'll be working with in this book.

1. Use File Explorer or Finder to find and open this app in your default browser:
   `\html_css_5\book_apps\ch07\town_hall\index.html`
2. Click on the link for Scott Sampson to see that page. This is the website that's presented at the end of chapter 7, but note that the Scott Sampson link is the only one that has been implemented.
3. Open this file in the Chrome browser.
   `\html_css_5\book_examples\ch15\06_animations\index.html`

   This is the example for figure 15-6 in chapter 15.

Exercise 1-3 Visit some Internet websites

Visit a small web site and look at the source code

1. Enter www.newtonforkranch.com in the address bar of your default browser and press the Enter key. That should display the home page for this website. Here, JavaScript is used to rotate the images at the top of the page.
2. Use the technique in figure 1-11 to view the source code for the home page. Here, the link elements with a rel attribute value of "stylesheet" identify the CSS files that do the formatting for the page, and the script elements identify the JavaScript files.
3. Click on styles.css in the href attribute in the second link element. That should open the first CSS file for this page. This shows how complicated the source code can be, even for a small website.

Visit other websites

4. Go to www.landsend.com, find a page like the one in figure 1-4, and experiment with the image swaps and rollovers. Those are done by JavaScript after all the images are loaded with the page.
5. View the HTML source code for that page and see how complicated the code can be for a large, eCommerce site. Then, view the code in one of the CSS files.
6. Visit other websites on your desktop computer and also on your mobile phone to see how they handle Responsive Web Design. Then, see if the sites provide web accessibility for the motor impaired by trying to do everything with just the keyboard, no mouse.
7. When you're through experimenting, close all the browser tabs that you've opened.

Chapter 2

How to code, test, and validate a web page

In this chapter, you'll learn how to create and edit HTML and CSS files. Then, you'll learn how to test those files to make sure they work correctly. When you're through with this chapter, you'll be ready to learn all the details of HTML and CSS coding.

Although you can use any text editor or IDE for web development with this book, this chapter also shows you how to use the text editor that we recommend. It is called *Visual Studio Code* (or just *VS Code*). It runs on Windows, macOS, and Linux. And it can help you work faster and better.

The HTML syntax

When you code an HTML document, you need to adhere to the rules for creating the HTML elements. These rules are referred to as the *syntax* of the language. In the four topics that follow, you'll learn the HTML syntax.

The basic structure of an HTML document

Figure 2-1 presents the basic structure of an *HTML document*. As you can see, every HTML document consists of two parts: the DOCTYPE declaration and the document tree.

When you use HTML5, you code the *DOCTYPE declaration* exactly as it's shown in this figure. It will be the first line of code in every HTML document that you create, and it tells the browser that the document is using HTML5. If you've developed web pages with earlier versions of HTML or XHTML, you will be pleased to see how much this declaration has been simplified.

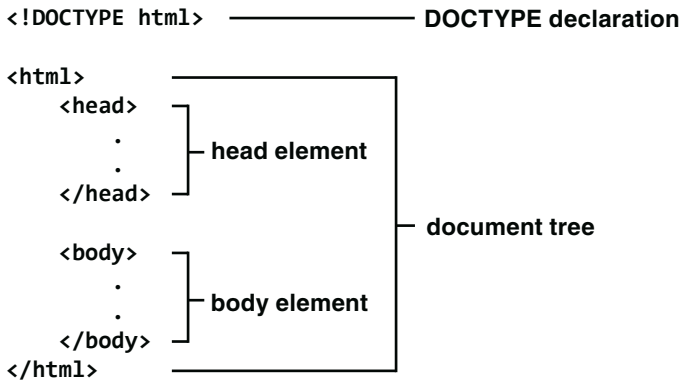
The *document tree* starts right after the DOCTYPE declaration. This tree consists of the *HTML elements* that define the web page. The first of these elements is the html element itself, which contains all of the other elements. This element can be referred to as the *root element* of the tree.

Within the html element, you should always code a head element and a body element. The head element contains elements that provide information about the page itself, while the body element contains the elements that provide the structure and content for the page. You'll learn how to code these elements in the next chapter.

You'll use the elements shown in this figure in every HTML document that you create. As a result, it's a good practice to start every HTML document from a template that contains this code or from another HTML document that's similar to the one you're going to create. Later in this chapter, you'll learn how you can use Visual Studio Code to do that.

When you use HTML, you can code elements using lowercase, uppercase, or mixed case. For consistency, though, we recommend that you use lowercase unless uppercase is required. The one exception we make is in the DOCTYPE declaration because DOCTYPE has historically been capitalized (although lowercase works too). You will see this use of capitalization in all of the examples and applications in this book.

The basic structure of an HTML document



A simple HTML document

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>San Joaquin Valley Town Hall</title>
    </head>
    <body>
        <h1>San Joaquin Valley Town Hall</h1>
        <p>Welcome to San Joaquin Valley Town Hall.</p>
        <p>We have some amazing speakers in store for you this season!</p>
        <p><a href="speakers.html">Speaker information</a></p>
    </body>
</html>
```

General coding recommendation for HTML

- Although you can code the HTML using lowercase, uppercase, or mixed case, we recommend that you do all coding in lowercase because it's easier to read.

Description

- An *HTML document* contains *HTML elements* that define the content and structure of a web page.
- Each HTML document consists of two parts: the DOCTYPE declaration and the document tree.
- The *DOCTYPE declaration* shown above indicates that the document is going to use HTML5. You'll code this declaration at the start of every HTML document.
- The *document tree* starts with the html element, which marks the beginning and end of the HTML code. This element can be referred to as the *root element* of the document.
- The html element always contains one head element that provides information about the document and one body element that provides the structure and content of the document.

Figure 2-1 The basic structure of an HTML document

How to code elements and tags

Figure 2-2 shows you how to code elements and tags. As you have already seen, most HTML elements start with an *opening tag* and end with a *closing tag* that is like the opening tag but has a slash within it. Thus, <h1> is the opening tag for a level-1 heading, and </h1> is the closing tag. Between those tags, you code the *content* of the element.

Some elements, however, have no content or closing tag. These tags are referred to as *empty tags*. For instance, the <br> tag is an empty tag that starts a new line, and the <img> tag is an empty tag that identifies an image that should be displayed.

The third set of examples in this figure shows the right way and the wrong way to code tags when one element is *nested* within another. In short, the tags for one element shouldn't overlap with the tags for another element. That is, you can't close the outer element before you close the inner element.

From this point on in this book, we will refer to elements by the code used in the opening tag. For instance, we will refer to head elements, h1 elements, and img elements. To prevent misreading, though, we will enclose single-letter element names in brackets. As a result, we will refer to <a> elements and <p> elements. We will also use brackets wherever else we think they will help prevent misreading.

Two elements with opening and closing tags

```
<h1>San Joaquin Valley Town Hall</h1>
<p>Here is a list of links:</p>
```

Two empty tags

```
<br>
<img src="logo.gif" alt="Murach Logo">
```

Correct and incorrect nesting of tags

Correct nesting

```
<p>Order your copy <i>today!</i></p>
```

Incorrect nesting

```
<p>Order your copy <i>today!</p></i>
```

Description

- Most HTML elements have an opening tag, content, and a closing tag. Each tag is coded within a set of brackets (<>).
- An element's *opening tag* includes the tag name. The *closing tag* includes the tag name preceded by a slash. And the *content* includes everything that appears between the opening and closing tags.
- Some HTML elements have no content. For example, the <br> element, which forces a line break, consists of just one tag. This type of tag is called an *empty tag*.
- HTML elements are commonly *nested*. To nest elements correctly, though, you must close an inner set of tags before closing the outer set of tags.

Figure 2-2 How to code elements and tags

How to code attributes

Figure 2-3 shows how to code the *attributes* for an HTML element. These attributes are coded within the opening tag of an element or within an empty tag. For each attribute, you code the attribute name, an equal sign, and the attribute value.

When you use HTML5, the attribute value doesn't have to be coded within quotation marks unless the value contains a space, but we recommend that you use quotation marks to enclose all values. Also, although you can use either double or single quotes, we recommend that you always use double quotes. That way, your code will have a consistent appearance that will help you avoid coding errors.

In the examples in this figure, you can see how one or more attributes can be coded. For instance, the second example is an opening tag with three attributes. By contrast, the third example is an empty img element that contains a src attribute that gives the name of the image file that should be displayed, plus an alt attribute that gives the text that should be displayed if the image file can't be found.

The next example illustrates the use of a *Boolean attribute*. A Boolean attribute can have just two values, which represent either on or off. To turn a Boolean attribute on, you code just the name of the attribute. In this example, the checked attribute turns that attribute on, which causes the related check box to be checked when it is rendered by the browser. If you want the attribute to be off when the page is rendered, you don't code the attribute.

The next set of examples illustrates the use of two attributes that are commonly used to identify HTML elements. The id attribute is used to uniquely identify just one element, so each id attribute must have a unique value. By contrast, the class attribute can be used to mark one or more elements, so the same value can be used for more than one class attribute. You'll see these attributes in a complete example in figure 2-6.

How to code an opening tag with attributes

An opening tag with one attribute

```
<a href="contact.html">
```

An opening tag with three attributes

```
<a href="contact.html" title="Click to Contact Us" class="nav_link">
```

How to code an empty tag with attributes

```
<img src="logo.gif" alt="Murach Logo">
```

How to code a Boolean attribute

```
<input type="checkbox" name="mailList" checked>
```

Two common attributes for identifying HTML elements

An opening tag with an id attribute

```
<div id="page">
```

An opening tag with a class attribute

```
<a href="contact.html" title="Click to Contact Us" class="nav_link">
```

Coding rules

- An attribute consists of the attribute name, an equal sign (=), and the value for the attribute.
- Attribute values don't have to be enclosed in quotes if they don't contain spaces.
- Attribute values must be enclosed in single or double quotes if they contain one or more spaces, but you can't mix the type of quotation mark used for a single value.
- Boolean attributes can be coded as just the attribute name. They don't have to include the equal sign and a value that's the same as the attribute name.
- To code multiple attributes, separate each attribute with a space.

Our coding recommendation

- For consistency, enclose all attribute values in double quotes.

Description

- *Attributes* can be coded within opening or empty tags to supply optional values.
- A *Boolean attribute* represents either an on or off value.
- The id attribute is used to identify a single HTML element, so its value can be used for just one HTML element.
- A class attribute with the same value can be used for more than one HTML element.

Figure 2-3 How to code attributes

How to code comments and whitespace

Figure 2-4 shows you how to code *comments*. Here, the starting and ending characters for two comments are highlighted. Then, everything within those characters, also highlighted here, is ignored when the page is rendered.

One common use of comments is to describe or explain portions of code. That is illustrated by the first comment.

Another common use of comments is to *comment out* a portion of the code. This is illustrated by the second comment. This is useful when you're testing a web page and you want to temporarily disable a portion of code that you're having trouble with. Then, after you test the rest of the code, you can remove the comment and test that portion of the code.

This figure also illustrates the use of *whitespace*, which consists of characters like tab characters, return characters, and extra spaces. For instance, the return character after the opening body tag and all of the spaces between that tag and the next tag are whitespace.

Since whitespace is ignored when an HTML document is rendered, you can use the whitespace characters to format your HTML so it is easier to read. In this figure, for example, you can see how whitespace has been used to indent and align the HTML elements.

That of course is a good coding practice, and you'll see that in all of the examples in this book. Note, however, that the code will work the same if all of the whitespace is removed. In fact, you could code all of the HTML for a document in a single line.

Although whitespace doesn't affect the way an HTML document is rendered, it does take up space in the HTML file. As a result, you shouldn't overdo your use of it. Just use enough to make your code easy to read.

An HTML document with comments and whitespace

```
<!DOCTYPE html>
<!--
    This document displays the home page
    for the website.
-->

<html>
    <head>
        <title>San Joaquin Valley Town Hall</title>
    </head>

    <body>
        <h1>San Joaquin Valley Town Hall</h1>
        <h2>Bringing cutting-edge speakers to the valley</h2>
    <!-- This comments out all of the HTML code in the unordered list
        <ul>
            <li>October: David Brancaccio</li>
            <li>November: Andrew Ross Sorkin</li>
            <li>January: Amy Chua</li>
            <li>February: Scott Sampson</li>
            <li>March: Carlos Eire</li>
            <li>April: Ronan Tynan</li>
        </ul>
    The code after the end of this comment is active -->
        <p>Contact us by phone at (559) 444-2180 for ticket information.</p>
    </body>
</html>
```

Our coding recommendations

- Use whitespace to indent lines of code and make them easier to read.
- Don't overdo your use of whitespace, because it does add to the size of the file.

Description

- An HTML *comment* is text that appears between the <!-- and --> characters. Since web browsers ignore comments, you can use them to describe or explain portions of your HTML code that might otherwise be confusing.
- You can also use comments to *comment out* elements that you don't want the browser to display. This can be useful when you're testing a web page.
- An HTML comment can be coded on a single line or it can span two or more lines.
- *Whitespace* consists of characters like tab characters, line return characters, and extra spaces.
- Since whitespace is ignored by browsers, you can use it to indent lines of code and separate elements from one another by putting them on separate lines. This is a good coding practice because it makes your code easier to read.

Figure 2-4 How to code comments and whitespace

The CSS syntax

Like HTML, CSS has a syntax that must be adhered to when you create a CSS file. This syntax is presented next.

How to code CSS style rules and comments

A CSS file consists of *style rules*. As the diagram in figure 2-5 shows, a style rule consists of a *selector* followed by a set of braces. Within the braces are one or more *declarations*, and each declaration consists of a *property* and a *value*. Note that the property is followed by a colon and the value is followed by a semicolon.

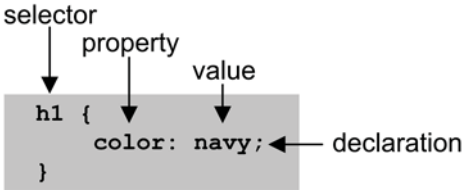
In this diagram, the selector is h1 so it applies to all h1 elements. Then, the style rule consists of a single property named color that is set to the color navy. The result is that the content of all h1 elements will be displayed in navy blue.

In the CSS code that follows, you can see four other style rules. Three of these contain only one declaration, but the third style rule consists of two declarations: one for the font-style property, and one for the border-bottom property.

Within a CSS file, you can also code comments that describe or explain what the CSS code is doing. For each comment, you start with /* and end with */, and anything between those characters is ignored. In the example in this figure, you can see how *CSS comments* can be coded on separate lines or after the lines that make up a style rule.

You can also use comments to comment out portions of code that you want disabled. This can be useful when you're testing your CSS code just as it is when you're testing your HTML code.

The parts of a CSS style rule



A simple CSS document with comments

```
/*********************************************************
* Description: Primary style sheet for valleytownhall.com
* Author:      Anne Boehm
*********************************************************/
/* Adjust the styles for the body */
body {
    background-color: #FACD8A;         /* This is a shade of orange. */
}

/* Adjust the styles for the headings */
h1 {
    color: #363636;
}
h2 {
    font-style: italic;
    border-bottom: 3px solid #EF9C00;  /* Adds a line below h2 headings */
}

/* Adjust the styles for the unordered list */
ul {
    list-style-type: square;           /* Changes the bullets to squares */
}
```

Description

- A CSS *style rule* consists of a selector and zero or more declarations enclosed in braces.
- A CSS *selector* consists of the identifiers that are coded at the beginning of the style rule.
- A CSS *declaration* consists of a *property*, a colon, a *value*, and a semicolon.
- To make your code easier to read, you can use spaces, indentation, and blank lines within a style rule.
- CSS *comments* begin with the characters /* and end with the characters */. A CSS comment can be coded on a single line, or it can span multiple lines.

Figure 2-5 How to code CSS style rules and comments

How to code basic selectors

The selector of a style rule identifies the HTML element or elements that the declarations should be applied to. To give you a better idea of how this works, figure 2-6 shows how to use the three basic selectors for CSS style rules. Then, in chapter 4, you'll learn how to code all types of selectors.

The first type of selector identifies HTML elements like body, h1, or <p> elements. For instance, the selectors in the first two examples apply to the body and h1 elements. These selectors are called *type selectors*.

The second type of selector starts with the hash character (#) and applies to the single HTML element that's identified by the id attribute. For instance, #copyright applies to the HTML element that has an id attribute with a value of copyright. As you can see, that's the last <p> element in the HTML code.

The third type of selector starts with a period (.) and applies to all of the HTML elements that are identified by the class attribute with the named value. For instance, .base_color applies to all elements with class attributes that have a value of base_color. In the HTML code, this includes the h1 element and the last <p> element.

Starting with chapter 4, you'll learn all of the coding details for style rules. But to give you an idea of what's going on in this example, here's a quick review of the code.

In the style rule for the body element, the font-family is set either to Arial (if the browser has access to that font) or the sans-serif type that is the default for the browser. This font is then used for all text that's displayed within the body element, unless it's overridden later on by some other style rule. So in this example, all of the text will be Arial or sans-serif, and you can see that font in the browser display.

The style rule for the body element also sets the font-size to 100% of the default size. Although this is the default, this selector is often coded for completeness. Next, the width of the body is set to 300 pixels, and the padding between the contents and the border is set to 1 em, which is the height of the default font.

In the style rule for the h1 element, the font-size is set to 180% of the default font size for the document (the size set by the selector for the body element). Then, in the style rule for the second <p> element (#copyright), the font size is set to 75% of the default font size, and the text is right-aligned. Here again, you can see how these style rules are applied in the browser display.

Last, in the style rule for the class named base_color, the color is set to blue. This means that both of the HTML elements that have that class name (the h1 element and the second <p> element) are displayed in blue.

This example shows how easy it is to identify the elements that you want to apply CSS formatting to. This also shows how the use of CSS separates the formatting from the content and structure that is defined by the HTML.

HTML elements that can be selected by element type, id, or class

```
<body>
    <h1 class="base_color">Student materials</h1>
    <p>Here are the links for the downloads:</p>
    <ul id="links">
        <li><a href="exercises.html">Exercises</a></li>
        <li><a href="solutions.html">Solutions</a></li>
    </ul>
    <p id="copyright" class="base_color">Copyright 2022</p>
</body>
```

CSS style rules that select by element type, id, and class

Type

```
body {
    font-family: Arial, sans-serif;
    font-size: 100%;
    width: 300px;
    padding: 1em;
}
h1 {
    font-size: 180%;
}
```

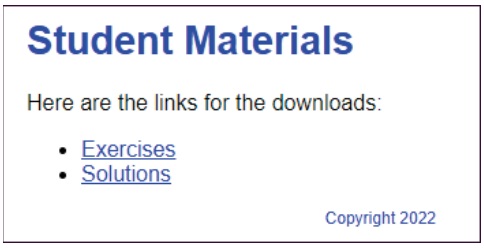
ID

```
#copyright {
    font-size: 75%;
    text-align: right;
}
```

Class

```
.base_color {
    color: blue;
}
```

The elements in a browser



Description

- To code a selector for an HTML element, you simply name the element. This is referred to as a *type selector*.
- If an element is coded with an id attribute, you can code a selector for that id by coding a hash character (#) followed by the id value, as in #copyright.
- If an element is coded with a class attribute, you can code a selector for that class by coding a period followed by the class name, as in .base_color.

Figure 2-6 How to code basic selectors

How to use VS Code to develop web pages

As you learned in chapter 1, you can use a text editor or an IDE to enter and edit the HTML and CSS files for a web application. The text editor that we recommend is *Visual Studio Code* (or *VS Code*), and the topics that follow present the basic skills for using it. Of course, if you prefer to use another editor or IDE, you can skip these topics.

How to work with folders

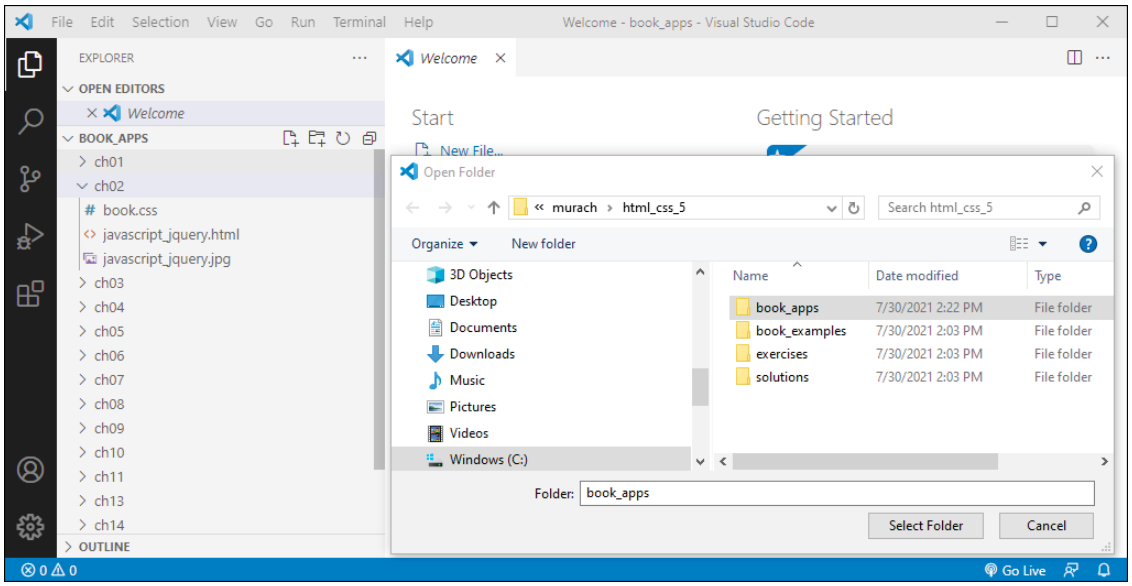
To work with a web application in VS Code, you start by opening the top-level folder for the application. That's the one that contains all the subfolders and files for the application. The first procedure in figure 2-7 shows how.

Instead of opening the top-level folder for an application, though, this example opens the book_apps folder that contains all the applications that are presented in this book. Also, the Explorer on the left side of VS Code shows the subfolders and files for the folder that's opened by the Open Folder dialog box. In other words, the Explorer shows what happens when the book_apps folder is opened.

After you open the top-level folder for an application, you can hide or display the folders and files in the Explorer by clicking on the folder names. That makes it easy to find the folders and files that you're looking for. Then, to add, rename, or delete a folder, you can use the procedures in the third group in this figure.

The last procedure in this figure shows how to set the color theme for VS Code. This sounds like a small item, but it can make a big difference. The theme we like is Light (Visual Studio), but you can select the one that works best for you.

The dialog box for choosing a top-level folder (book_apps) in VS Code



How to open the top-level folder for an application

- Select File→Open Folder from the menu system.
- Use the resulting dialog box to select the folder and click Select Folder. This sets up the folder and file tree that's shown in the Explorer.

How to hide or display the folders and files in the Explorer

- Click on the folder names.

How to add, rename, or delete a folder in the Explorer

- To add a subfolder to a folder, select the folder in the Explorer and click the New Folder icon that's displayed to the right of the top folder.
- To rename a folder, right-click on it and select Rename. Then, edit the name.
- To delete a folder, right-click on it and select Delete.

How to set the color theme for VS Code

- Select File→Preferences→Color Theme from the menu bar. (The examples in this chapter use the Light (Visual Studio) theme.)

Description

- Please refer to appendix A to learn how to install VS Code on a Windows or macOS system.

Figure 2-7 How to work with folders in VS Code

How to work with files

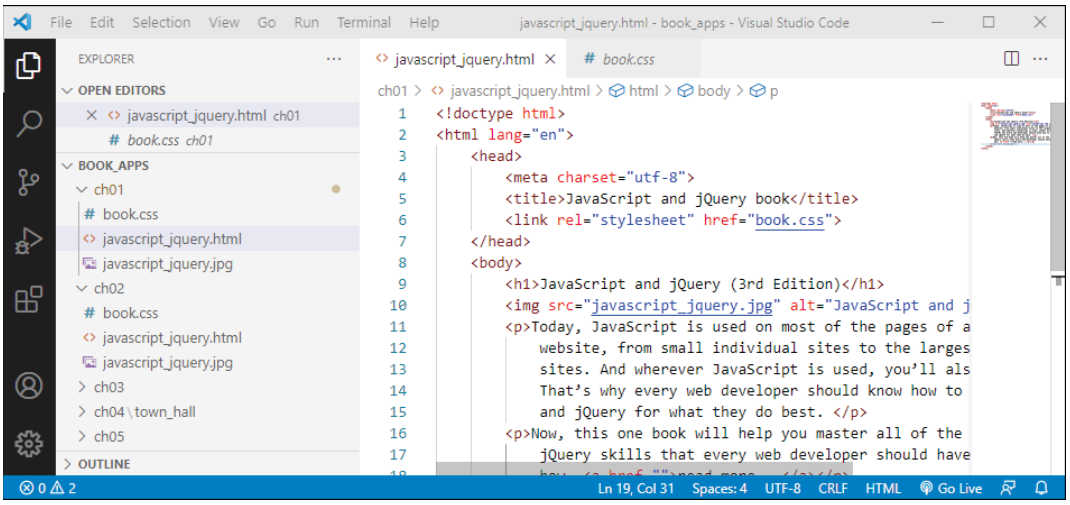
Figure 2-8 presents the basic skills for working with files in VS Code. To *open* a file, you find it in the Explorer and double-click on it. Then, it's displayed in a tab in the main window. This is called Standard Mode, and you use it when you're going to edit a file.

In contrast, to *preview* a file, you just click on it (no double-click). Then, the name of the file is displayed in italics on the tab, and the tab is re-used if you open or preview another file. This is called Preview Mode, and you use it when you want to take a quick look at a file or make just a few changes to it.

To save, close, add, rename, and delete files, you can use the other procedures in this figure. They work the way they do with most applications. But note that when you add a new file to a folder, you need to give it a name with either the html or css extension. Then, VS Code will know what type of file it is, and the editor will expect the syntax for that type of file.

If you've been experimenting with VS Code, you may have noticed that a Welcome page is displayed whenever you start VS Code after you've closed it with no files open or in preview mode. Now, if you want to stop this page from being displayed, you can use the last procedure in this figure.

VS Code with files in Standard and Preview mode



How to open or preview a file

- To *open* a file, *double-click* on it in the Explorer. This displays the file in a tab in the editor with the name of the file in normal font style, indicating that it's in Standard Mode.
- To *preview* a file, *click* on it in the Explorer. This displays the file in a tab in the editor with the name of the file in italics, indicating that it's in Preview Mode. If you open or preview another file, VS Code reuses the tab.

How to save or close a file

- If you want to save the changes to a file without closing it, select File→Save.
- To save the changes to more than one file, select File→Save All or click the Save All icon that shows when you point to Open Editors in the Explorer.
- To close a file, click the X in the upper right corner of the tab for the file, or click the X to the left of the file name in the Open Editors list.
- If you try to close a file that has been changed but not saved, you'll be asked if you want to save the changes.

How to add a new file to a folder

- Select the folder in the Explorer and click the New File icon that's displayed to the right of the top folder. Then, enter the name for the file and be sure to include the html or css extension.

How rename or delete a file

- To rename a file, right-click on it and select Rename. Then, edit the name.
- To delete a file, right-click on it and select Delete.

How to stop the Welcome page from being displayed

- Scroll down to the bottom of the page, and uncheck the check box.

Figure 2-8 How to work with files in VS Code

How to edit an HTML file

Figure 2-9 shows how to edit an HTML file with VS Code. When you open a file with an html extension, VS Code knows what type of file you're working with so it can use color to highlight the syntax components. It also uses IntelliSense to provide *completion lists* that let you select the names of elements and attributes as you type. And when you enter an opening tag, it automatically adds the closing tag.

To display the completion list for an element, you type a left bracket (<). This displays a list of all the available elements. Then, if you type one or more letters, the list is filtered to just the ones that start with those characters. In this figure, for example, I typed the letter "i" so only the elements that start with that letter are displayed.

You can use a similar technique to enter attributes within an opening tag. To start, enter a space and one or more letters after the element name. That displays a list of all the attributes that start with those letters. Then, when you select an attribute, VS Code inserts the attribute along with an equal sign and double quotes, and you can enter the value of the attribute between the quotes.

When you add a new file to a folder, you should realize that VS Code doesn't generate any code for it. However, you can generate the starting code for an HTML file by entering an exclamation mark (!) and pressing the Tab key. This is a feature of Emmet, which is a third-party plugin that's used by VS Code.

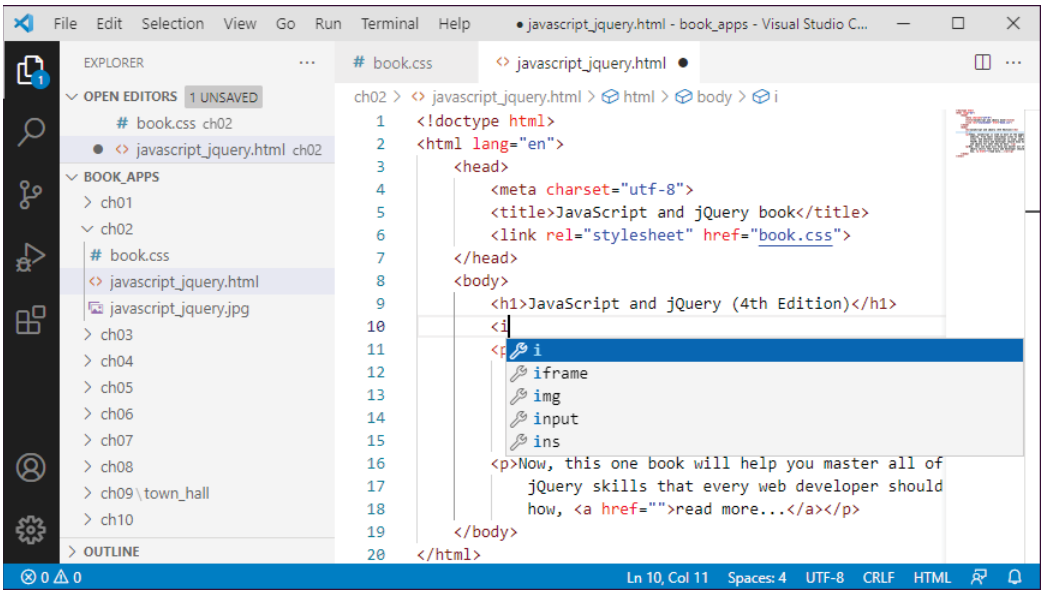
Because the Emmet starting code is minimal, another option is to copy code from another file and paste it into the new file. Or, you can open a similar file and then save it with a new name. Then, you can delete the code you don't need and add the code you do need.

As you become more familiar with HTML, you can take this to the next level by creating HTML files that contain the elements you need for different types of pages. Then, you can use those files as *templates* for creating new pages. To do that, you can open the template file and save it with a new name before you modify the file. That way, the original template file remains unchanged. What you don't want to do is to start new files from scratch.

Although this figure doesn't illustrate it, you should know that you can split the editor so you can view two or more files at once. That can be useful when you want to copy code from one file to another or compare the code in two files. The last procedure in this figure shows just one of the ways that you can split the editor. It uses View→Editor Layout to split the editor into two or more columns or rows. Then, you can open a different file in each of the rows and columns.

Although it takes some experimentation to master the use of split editors, it's worth the effort. To help you get started, exercise 2-2 guides you through the basic skills for using this feature.

The completion list for selecting an HTML element



Common coding errors

- An opening tag without a closing tag.
- Misspelled element or attribute names.
- Quotation marks that aren't paired.
- Incorrect file references in link, img, or <a> elements.

How to use the IntelliSense feature

- IntelliSense displays *completion lists* for elements and attributes. To insert an item from a completion list, click on it or highlight it and press the Tab or Enter key.
- When you enter a bracket at the end of an opening tag, VS Code enters the closing tag.

How to add starting code to a new file

- Enter an exclamation mark (!) into the file and press the Tab key.

How to split the screen so you can work with two or more files at once

- After you open one file, use View→Editor Layout to split the editor into two or more columns or rows. Then, open a second file in one of the columns or rows.

Description

- VS Code provides features for editing HTML files like color coding and IntelliSense. It also makes it easy for you to add the starting code for a file.

Figure 2-9 How to edit an HTML file in VS Code

How to use the HTMLHint extension to find HTML errors

Because VS Code doesn't provide for HTML error checking out of the box, we recommend that you install an extension to VS Code that checks your HTML code as you enter it. One popular extension is the HTMLHint extension developed by Mike Kaufman, and figure 2-10 shows how to install and use it.

To start, you click the Extensions icon in the left bar. Then, the Extensions window is displayed in place of the Explorer window. This window lists any extensions that are already installed as well as any recommended extensions.

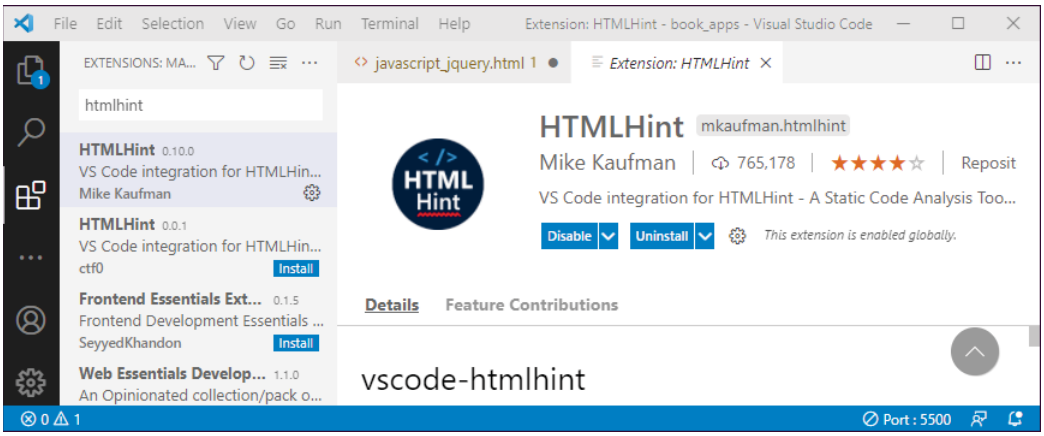
To find the HTMLHint extension, you can type "htmlhint" in the text box at the top of the Extensions window as shown in the first screen in this figure. When you do that, you'll notice that there are two extensions with that name. We recommend that you install the one developed by Mike Kaufman.

After you install the HTMLHint extension, VS Code checks your code as you enter it. Then, if it detects an error, it displays a wavy line under the error as shown in the second screen in this figure. Here, "<link" is underlined because the link tag doesn't end with a right bracket.

To display the description of an error, you can hover the mouse over the wavy underline. Alternatively, you can open the Problems window to display a list of all the errors. Then, you can click on an error to display it in the file. The Explorer also indicates whether a file contains errors by displaying the number of errors to the right of the filename.

Although HTMLHint can identify syntax errors, it has no way of knowing what the correct code should be for file references in link, img, or <a> elements. As a result, you must discover those errors when you test the web page. If, for example, the file reference for a style sheet in a link element is incorrect, the CSS won't be applied. If the file reference in an img element is incorrect, the image won't be displayed. And if the file reference for an <a> element is incorrect, the browser won't access the correct page.

VS Code as the HTMLHint extension is installed



The Problems window with an error displayed

How to install the HTMLHint extension

1. Click the Extensions icon in the left bar, and enter “htmlhint” in the text box at the top of the window to filter the available extensions.
2. Click the Install button for the HTMLHint extension from Mike Kaufman.

How to identify the errors that are marked by HTMLHint

- If HTMLHint detects an HTML syntax error, it underlines it with a wavy line.
- To get the description for an error, hover the mouse over the wavy line.
- To see all the errors in a file, you can display the Problems window (View→Problems). Then, you can click on an error to take you to it in the file.

Description

- The HTMLHint extension checks your code as you enter it so you can correct your errors right away.

Figure 2-10 How to use the HTMLHint extension to find HTML errors

How to edit a CSS file

When you open a file with the css extension, VS Code knows what type of file you're working with. That way, it can use color to highlight the syntax components. It can use IntelliSense to provide *completion lists* that let you select the names of elements, properties, and values. And when you enter an opening brace, it automatically adds the closing brace.

If you start to enter the name of an element for a CSS style rule, for example, VS Code displays a list of the elements with the letters you enter. If you enter one or more letters to start a property declaration, VS Code displays a list of the properties that start with those letters. And if you start an entry for a property value, VS Code displays a list of possible values.

The example in figure 2-11 shows how completion lists work with CSS. Here, I entered a selector for the h2 element, followed by a space and a left brace, and VS Code added the right brace. Then, after I pressed Enter to start a new line and I entered the letter "b" for the first property, the completion list displayed all of the properties that start with that letter.

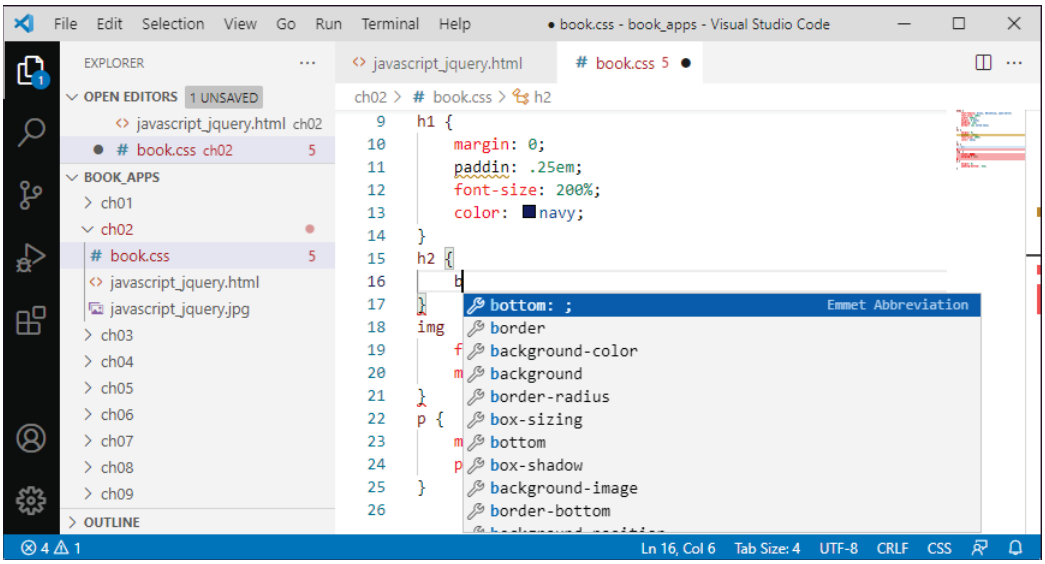
Similarly, when you enter the name of a property, VS Code enters the colon that follows the property as well as the semi-colon at the end of the property declaration. Then, you just need to enter the value for the property.

Unlike HTML files, VS Code provides error checking for CSS files out of the box. If it detects an error as you type, it displays a wavy line under the error. In this example, the padding property of the style rule for the h1 element is underlined because it's spelled incorrectly. To display the description of an error, you can hover the mouse over the wavy underline. And you can open the Problems window to display a list of all the errors. Then, you can click on an error to display it in the file.

The Explorer also indicates whether a file contains errors. To do that, it displays the number of errors in the file to the right of the filename. In this case, five errors have been detected.

Here again, you may want to split the editor when you work with CSS files. That can make it easier to copy code from one file to another. You can also split the editor with the HTML file in one editor and the related CSS file in another editor. That makes it easy to see the relationships between the two.

The completion list for selecting a CSS property



Common coding errors

- Braces that aren't paired correctly.
- Misspelled property names.
- Missing semicolons.
- Id or class names that don't match the names used in the HTML.

How to use the IntelliSense feature

- IntelliSense displays *completion lists* for elements, properties, and values. To insert an item from a completion list, click on it or highlight it and press the Tab or Enter key.
- When you select a property from a completion list, VS Code adds the colon and the semicolon for the declaration. Then, you just need to enter or select the value.
- When you enter the left brace for a style rule, VS Code adds the right brace.

How to identify the errors that are marked by VS Code

- If VS Code detects a CSS syntax error, it underlines it with a wavy line.
- To get the description for an error, hover the mouse over the wavy line.
- To see all the errors in a file, you can display the Problems window (View→Problems). Then, you can click on an error to take you to it in the file.

Description

- VS Code provides features for editing CSS files like color coding and IntelliSense. VS Code also identifies CSS syntax errors.

Figure 2-11 How to edit a CSS file in VS Code

How to use the Live Server extension to open an HTML file in a browser

As you develop web applications, you'll want to open the HTML files in a browser so you can see how they look and how they work. Unfortunately, VS Code doesn't provide an easy way to do that out of the box. Because of that, we recommend that you install an extension that does make it easy.

One popular extension for doing that is Live Server, and figure 2-12 shows how to install it. Once installed, you can click the Go Live icon at the bottom of VS Code to launch a local development server and open the active file in the default browser. In this example, the default browser is Chrome.

Once that's done, you can make changes to the HTML or CSS code for the page, save the changes, and the browser page is automatically updated. You don't have to refresh the page.

To open a different HTML file in the browser from the same development server, you can right-click on its file name in the Explorer and select Open with Live Server. That displays the new page in the same browser, but in a new tab of that browser.

While the local development server is open, the Go Live icon is replaced with an icon that has the server port number on it. Then, you can click on this icon to close the server. Note, however, that this doesn't close the browser window, so you'll eventually want to do that.

If you don't install an extension like Live Server, you should know that VS Code provides another way to open a file in a browser. Just right-click on the file you want to open in the Explorer and select Reveal in File Explorer (Windows) or Reveal in Finder (macOS). Then, the folder that contains the file will be displayed in the File Explorer or Finder, and you can double-click on the file to open it in your default browser.

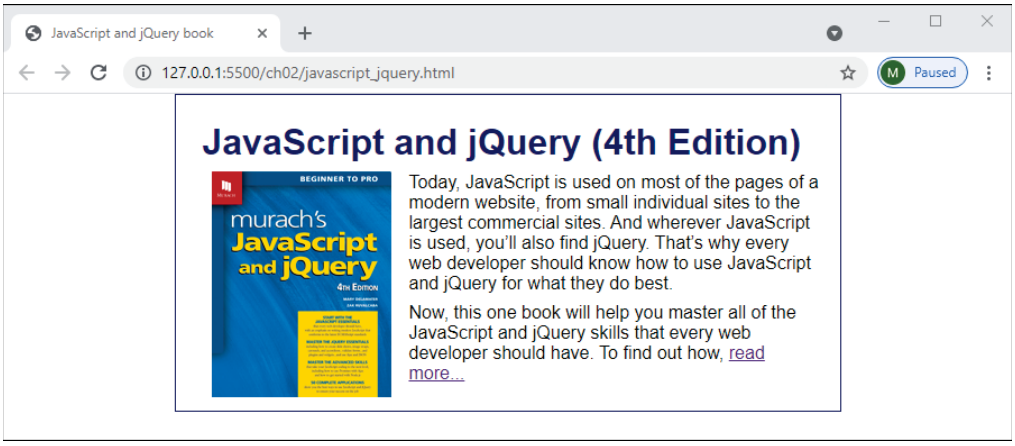
Every time you open an HTML file from the File Explorer or Finder, though, a new browser or browser tab is opened. To avoid that, you can open the file in the browser just once. Then, after you use VS Code to fix the errors, you can save the changes, switch to the browser, and click on the Reload or Refresh icon to reload the file with the changes.

How install the Live Server extension

1. Click the Extensions icon in the left bar, and enter “live server ritwick dey” in the text box to find the version of Live Server that we recommend.
2. Click the Install button for the Live Server extension.

Click the Go Live icon to open an HTML file on a development server

The web page on the development server in the default Chrome browser



How to open an HTML file in the default browser

- To open the active editor file, click the Go Live icon.
- To open any file in the Explorer, right-click on it and select Open with Live Server.

Description

- When you open an HTML file with Live Server, the Go Live icon is replaced with the port number for the server, which you can click when you want to close the server.
- After you make changes to your HTML or CSS code, you can save them and switch to the browser to view the changes without having to click on the Refresh or Reload icon.

Figure 2-12 How to use the Live Server extension to open an HTML file in a browser

How to test and debug a web page

When you *test* a web application, your goal is to find all the errors. When you *debug* an application, your goal is to fix all the errors. The topics that follow present some ideas for testing and debugging.

How to test a web page

To test a web page, you start by running it. To do that, you can use one of the ways shown in figure 2-13. Then, to test the web page, you check its contents and appearance to make sure they're exactly the way you want them. You also check each link to make sure it goes where it's supposed to go. If you find errors, you should note them and then debug them.

When the web page works right in one browser, you'll want to test it in the other browsers that are in common use. An easy way to do that is to copy the URL from one browser, open another browser, and paste the URL into that browser.

When you test a web page in more than one browser, you will sometimes find that the page works on one browser, but doesn't work on another. That's usually because one of the browsers makes some assumptions that the other browser doesn't. If, for example, you have a slight coding error in an HTML file, one browser might make an assumption that fixes the problem, while the other doesn't.

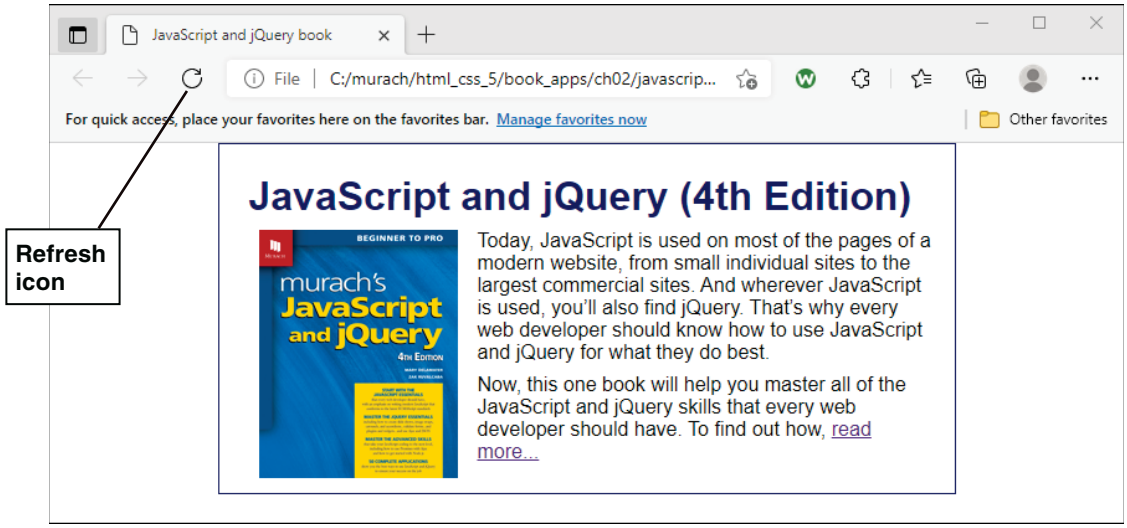
How to debug a web page

To debug a web page, you find the causes of the errors and correct them. Often, the changes you make as you debug a web page are just minor adjustments or improvements. But sometimes, the web page doesn't look at all the way you expected it to. Often, these errors are caused by trivial coding problems like missing tags, quotation marks, and braces, but finding these problems can be hard to do when your files consist of dozens of lines of code.

To help you find the causes of your errors, you can use the Developer Tools for your browser. As figure 4-17 in chapter 4 shows, these tools make it easy to see how the HTML and CSS are related.

When you test a page for the second time, you don't need to run it again. Instead, after you save the changes, you can click on the Refresh or Reload icon in the browser's toolbar. But you don't need to do that if you're using VS Code with the Live Server extension because your changes are applied in the browser as soon as you save them.

The HTML file in the Edge browser



Two ways to run a web page that's on your computer or local network

- Use the features of your text editor or IDE, like Live Server with VS Code.
- Find the file in your file explorer. Then, double-click on it to open it in your default browser. Or, right-click on it and use the Open With command to select the browser.

How to test a web page

- Check the contents and appearance of each page.
- Click on all of the links on each page to make sure they work properly.
- After you've tested your web pages on one browser, test them on the other browsers that your users may be using.

How to debug a web page

- Find the causes of the errors, fix the errors, and save the changes.
- To find the causes of your errors, you can use Chrome's Developer Tools, as shown in figure 4-17 of chapter 4. This tool shows you how the CSS relates to the HTML.
- After you find the cause and fix it, you test the page again. To do that, you can switch to the browser from your text editor or IDE and then click the Refresh or Reload icon.

Description

- When you *test* a web page, you try to find all the errors.
- When you *debug* a web page, you fix the errors and test again.

Figure 2-13 How to test and debug a web page

How to validate HTML and CSS files

After you test and debug your web pages, you're pretty sure that the HTML and CSS is valid. But you can take validation to the next level by using an official validation service to *validate* your files.

These services are useful when a file is large, the HTML or CSS isn't working right, and you can't spot any errors. HTML and CSS validation may also ensure that your code will work on an infrequently-used browser that isn't one of the browsers that you're using for testing.

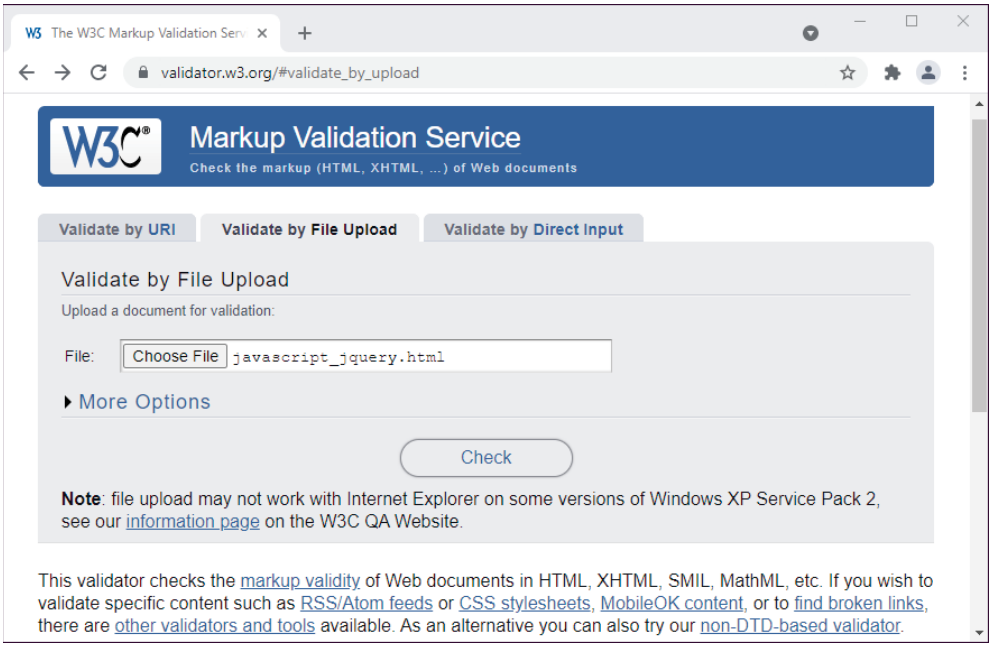
How to validate an HTML file

One of the most popular websites for validating HTML is the one for the W3C Markup Validation Service that's shown in figure 2-14. When you use this website, you can provide the HTML code that you want to validate in the three ways shown in this figure.

In this example, the Validate by File Upload tab is shown. Then, you click the Browse button to find the file that you want to validate. Once that's done, you click the Check button to validate the document.

If the HTML code is valid when a document is validated, the validator displays a message to that effect. However, it may also display one or more warning messages. On the other hand, if the code contains errors, the validator will display a list of the errors. Then, you can use the error messages to help you correct your code.

The home page for the W3C Markup Validation Service



How to use the W3C Markup Validation Service

- Go to the URL that follows, identify the file to be validated, and click the Check button:

  https://validator.w3.org/

Three ways to provide the code to be validated

- If the file you want to validate has been uploaded to a web server, you can enter its URL on the Validate by URI tab.
- If the file hasn't been uploaded to a web server, you can locate it on the Validate by File Upload tab.
- You can also copy and paste the HTML code into the Validate by Direct Input tab.

Description

- To *validate* the HTML for a page, you can use a program or website for that purpose. One of the most popular websites is the W3C Markup Validation Service.
- Validation insures that your code is correct, which may improve SEO. Validation can also help you find errors in your HTML that you aren't aware of.
- If the HTML document is valid, the validator will indicate that the document passed the validation. However, one or more warnings may still be displayed.
- If the HTML document isn't valid, the validator will list and describe each error and warning. You can use the Message Filtering button above the list to display a count of the errors and warnings and to select which errors and warnings you want displayed.

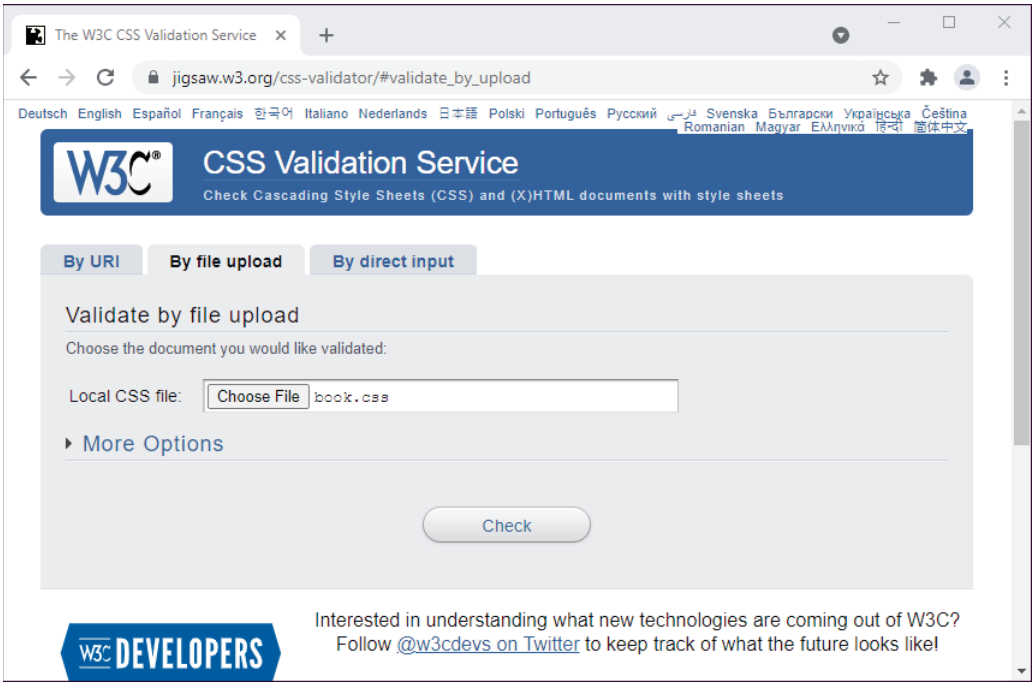
Figure 2-14 How to validate an HTML file

How to validate a CSS file

You can validate a CSS file the same way you validate an HTML file. But this time, you use a website like the W3C CSS Validation Service. Just use the link in figure 2-15 to access this service. Then, you can use any one of the three tabs to validate the CSS file.

If the file contains errors when it is validated, the validation service displays a list of the errors. Then, you can use the error messages to identify and correct your coding errors.

The home page for the W3C CSS Validation Service



How to use the W3C CSS Validation Service

- Go to the URL that follows, identify the file to be validated, and click the Check button:

  https://jigsaw.w3.org/css-validator/

Description

- To validate the CSS for a page, you can use a program or website for that purpose. One of the most popular websites is the W3C CSS Validation Service.
- Validation not only insures that your code is correct, but it can also help you find errors in your CSS that you aren't aware of.

Figure 2-15 How to validate a CSS file

Perspective

Now that you've completed this chapter, you should be able to create and edit HTML and CSS files using Visual Studio Code. Then, you should be able to test those files by displaying their web pages in your default web browser or any of the other browsers. You should also be able to validate the HTML and CSS files for a web page whenever that's necessary.

At this point, you're ready to learn the coding details for HTML and CSS. So, in the next chapter, you'll learn the details for coding the HTML elements that define the structure and content for a web page. And in chapters 4, 5, and 6, you'll learn the details for coding the CSS style rules that format the HTML content.

Terms

syntax
HTML document
DOCTYPE declaration
document tree
HTML element
root element
opening tag
closing tag
content of an element
empty tag
nested elements
attribute
Boolean attribute
comment
comment out
whitespace
style rule
selector
declaration
property
value
type selector
Visual Studio Code (VS Code)
preview a file
completion list
template
testing
debugging
HTML validation
CSS validation

Summary

- An *HTML document* consists of a *DOCTYPE declaration* that indicates what version of HTML is being used and a *document tree* that contains the *HTML elements* that define the web page.
- The *root element* in a document tree is the html element, which always contains a head element and a body element. The head element provides information about the page, and the body element provides the structure and content for the page.
- Most HTML elements consist of an *opening tag* and a *closing tag* with *content* between these tags. When you *nest* elements with HTML, the inner set of tags must be closed before the outer set.
- *Attributes* can be coded in an opening tag to supply optional values. An attribute consists of the name of the attribute, an equal sign, and the attribute value. To code multiple attributes, you separate them with spaces.
- An *HTML comment* can be used to describe or explain a portion of code. Because comments are ignored, you can also use comments to *comment out* a portion of HTML code so it isn't rendered by the browser. This can be helpful when you're testing your HTML code.
- *Whitespace* consists of characters like tab characters, line return characters, and extra spaces that are ignored by browsers. As a result, you can use whitespace to indent and align your code.
- A *CSS style rule* consists of a selector and declarations. The *selector* identifies the HTML elements that are going to be formatted. Three of the common CSS selectors select by element (called a *type selector*), ID, and class.
- A *declaration* in a CSS style rule consists of a *property*, a colon, a *value*, and a semicolon.
- *CSS comments* work like HTML comments. However, CSS comments start with /* and end with */, and HTML comments start with <!-- and end with -->.
- *Visual Studio Code*, or just *VS Code*, is a text editor that can be used to edit HTML and CSS code. To help you enter code, VS Code provides features like color coding and Intellisense.
- Although VS code provides syntax checking for CSS code, you need to add an extension like HTMLHint to provide syntax checking for HTML code.
- To make it easy to display HTML files in a browser from VS Code, you need to install an extension like Live Server.
- To *test* an HTML file, you run it on all the browsers that your clients may use. When you discover problems, you *debug* the code and test it again.
- To *validate* an HTML or CSS file, you can use a program or website for that purpose. Sometimes, that can help solve hard-to-detect debugging problems.

Before you do the exercises for this book...

If you haven't already done it, you should install the Chrome browser and the applications, examples, and exercises for this book. If you're going to use VS Code as your text editor, you should also download and install that editor. The procedures for doing that for both Windows and macOS users are in appendix A.

Exercise 2-1 Get started right with VS Code

If you're going to be using VS Code as your text editor, this exercise will get you started right. If you aren't going to use VS Code, you can do exercise 2-2.

Start VS Code, review the Welcome page, and change the color theme

1. Start VS Code. Then, review the Welcome page that's in the editor window to see what you can do from it. You don't really need it, though, so you can close that page. And if you don't want it to be opened any more, you can uncheck the box at the bottom of the Welcome page.
2. Select File→Preferences→Color Theme if you're using Windows or Code→Preferences→Color Theme if you're using macOS. Then, review the available color themes. If you want to change the theme, do that now.

Open the folders for this book

3. Use the procedure in figure 2-7 to open the book applications that are stored in this folder:

   ```
   murach\html_css_5\book_apps
   ```

   After you open this folder, you should see the book_apps folder and its subfolders in the Explorer window.
4. Use the same procedure to open this exercises folder:

   ```
   murach\html_css_5\exercises
   ```

   When you do that, note that the exercises folder replaced the book_apps folder.

Edit the code in a book application and use Live Server to test it

5. In the Explorer window, click on the ch02 folder to display the files in that folder. Then, double-click on the file named javascript_jquery.html to open that file. Note that it is now listed under Open Editors.
6. Install the Live Server extension as shown in figure 2-12. Then, click on the Go Live icon to open the HTML file in your default browser.
7. In the javascript_jquery.html file in VS Code, start a new line after the second <p> element. Then, type <p in that line to start a new element and note how VS Code provides a completion list. Select just the p from this list, type the closing bracket for the element, and note how VS code adds the closing tag.

8. Add this text to the <p> element that you just created:

```
For customer service, call us at 1-555-555-5555.
```

   Then, save this change. Now, look at the file in the browser that you opened in step 6 to see that it has been updated.

9. Double-click on the file named book.css to open that file. Then, change the color property for the h1 element to red, save the change, and look at the browser to see that the heading is now displayed in red.

Use the split editor feature

10. Close the book.css file so just the javascript_jquery.html file is open. Then, use View→Editor Layout→Two Columns to split the editor. That will open up a second editor window with nothing in it, and two groups will be shown under Open Editors.
11. Click in the right editor to put the cursor there, and double-click on the book.css file in the Explorer window. This will open the CSS file in that second editor window. Now, you can see the code for the HTML and CSS files at the same time.
12. In the CSS file, change the float property for the img element from left to right. Now, save this change, and note the change in the browser.
13. Close the CSS file, and note that the editor returns to one column. Then, close the HTML file so no files are open.

Start new HTML files

14. Use File→New File to start a new file, and use File→Save As to save it with the name testpage1.html in the ch02 folder. Then, enter an exclamation mark into the new file and press the Tab or Enter key.

    This should insert the starting code for an HTML document into the file. That's one way to start a new HTML file, but note that the starting code is minimal.

15. Open the javascript_jquery.html page in the ch02 folder, and save it as testpage2.html. This is another way to start a new file. Now, you just delete what you don't want, change what needs to be changed, and add what needs to be added.

    For example, (1) change the text in the title tag to "HTML and CSS"; (2) change the text in the H1 tag to "HTML and CSS (5th Edition)"; (3) change the code for the src attribute in the image tag to "html_css_5.jpg" and change the alt attribute to "HTML and CSS"; (4) change the text in the first <p> element to "Marketing copy goes here"; and (5) delete the second <p> element.

    Now, save the file. Then, to open this file in a browser, right-click on the file in the VS Code Explorer and select Open with Live Server.

Experiment and clean up

16. If you want to experiment, please do that. When you're through, move the cursor to a file in the Open Editors list and click the X on its left to close one file at a time.

17. When all of the files have been closed, close the browser and close VS Code.

Exercise 2-2 Edit a web page with any editor or IDE

If you aren't using VS Code, this exercise will guide you through the process of editing a web page with any text editor or IDE.

Open and test the files for the book page

1. Start your text editor. Then, open the HTML file named javascript_jquery.html that's in this folder:
   `murach\html_css_5\exercises\ch02`
2. Still in the editor, open the CSS file named book.css that's in the same folder.
3. Run the HTML file in Chrome.

Modify the HTML and CSS code and test again

4. Go to the javascript_jquery.html file, and add a <p> element at the bottom of the page that has this content:
   `For customer service, call us at 1-555-555-5555.`
   Then, save and test this change in your browser.
5. Go to your text editor and display the book.css file. Then, change the color property for the h1 element to red, save the change, and test it in your browser.
6. Still in the CSS file, change the float property for the img element from left to right. Then, save and test that change.
7. Continue to experiment on your own, and close the files when you're done.

Exercise 2-3 Validate HTML and CSS files

You can do this exercise whether or not you're using VS Code as your editor.

1. Use your editor or IDE to open the HTML file named javascript_jquery.html that's in this folder:
   `murach\html_css_5\exercises\ch02`
   Then, delete the ending > for the img tag, and save the file.
2. Go to the site in figure 2-14, and use the Validate by File Upload tab to validate the file. Then, scroll down the page to see the 3 error messages that are displayed for the one error.
3. Open the CSS file named book.css that's in the step 1 folder. Then, delete the semicolon after the font-size property in the h1 style rule, and save the file.
4. Go to the site in figure 2-15, and use the By File Upload tab to validate the file. This time, you'll see 1 error message.
5. Fix the errors, save the files, and validate the files again so there are no errors.

Chapter 3

How to use HTML to structure a web page

In chapter 2, you saw the basic structure of an HTML document, and you learned the basic techniques for coding the elements that make up a document. Now, in this chapter, you'll learn how to code the HTML elements that you'll use in most of your documents. Then, in the next three chapters, you'll learn how to use CSS to format those HTML elements.

How to code the head section

The head section of an HTML document contains elements that provide information about the web page rather than the content of the page. It requires a minimum amount of code, and figure 3-1 shows how to code it.

How to include metadata

You should use the meta element shown in this figure to provide two items of *metadata* for every web page. In the example, the first meta element specifies the character encoding used for the page, and UTF-8 is the encoding that's commonly used for the World Wide Web.

The second meta element provides a description that can be used by search engines to index the page. This description should summarize the contents of the web page. And when it's shown in the search results, it should encourage users to access your site.

How to code the title element and link to a favicon

The head section of every web page should also include a title element that describes the content of the page. In the HTML in this figure, you can see that this element gives the name of the organization followed by the keywords *speakers* and *luncheons*. This title is used by search engines, and it appears in the search results to help the users decide whether they want to go to that page.

The content of the title element is also displayed in the browser's tab for the page. As you can see in this figure, only the portion of the title that fits on the tab is displayed. But if the user hovers the mouse over the tab, the entire title should be displayed in a tooltip.

The link element can be used in the head section to link a custom icon, called a *favicon*, to the web page. This causes the icon to be displayed to the left of the title in the tab for the page. But note that you don't need to code this element when you deploy the website to an Internet server. Instead, you can place the favicon in the root folder for the website and it will automatically appear in the tab for each page.

A browser that shows the title and favicon

A head section that sets a title, a favicon, and two items of metadata

```
<head>
    <meta charset="utf-8">
    <meta name="description" content="A yearly lecture series with speakers
        that present new information on a wide range of subjects">
    <title>San Joaquin Valley Town Hall | speakers and luncheons</title>
    <link rel="shortcut icon" href="favicon.ico">
</head>
```

SEO guidelines for the metadata

- Code description metadata that summarizes the contents of the page. The description should be unique for each page, and it should encourage uses to click on the link when it's displayed in search results.

SEO guidelines for the title tag

- Code a title tag in the head section of each web page. It should describe the page's content, and it should include the one or two keywords, called *focus keywords*, that you want to be used to rank the page.
- The title should entice the reader to click on it when it's shown in the search results for a search engine. It should be unique for each web page, and it should be limited to around 65 characters because most search engines don't display more than that in their results.

Description

- The meta element provides information about the HTML document that's called *metadata*. It should be used to provide the charset and description for each page.
- The title element specifies the text that's displayed in the browser's tab for the web page. It is also used as the name of a favorite or bookmark for the page.
- A *favicon* is an icon that appears to the left of the title in the browser's tab for the page. It may also appear to the left of the URL in the browser's address bar, and it may be used in a favorite or bookmark.
- To specify a favicon for a page, you use favicon.ico as the name of the favicon and you code a link element exactly like the one above. Then, the favicon will be displayed when you test the page on your computer or local server.
- Note, however, that you don't need to code the link element for the favicon when you deploy the website to an Internet server. You just need to store the favicon in the root folder.

Figure 3-1 How to code the head section of an HTML document

How to present the contents of a web page

The next three figures show how to present the contents of a web page. That includes how to structure that content for web accessibility and search engine optimization.

How to code the lang attribute

Figure 3-2 starts by showing how to code the lang attribute that should identify the language for each of your web pages. To do that, you code the lang attribute for the html element of each page just as it's shown. In this case, that attribute is set to "en" so English is the language.

How to code headings and paragraphs

Headings and paragraphs provide most of the text for a web page. They are defined by the HTML elements in the second table in figure 3-2. These elements are *block elements*, which means that they start on a new line when they are displayed.

In the example in this figure, you can see how these elements work. Here, the HTML uses the h1, h2, and <p> elements to generate the text that's shown. When these elements are displayed by a browser, each element has a default font and size that's determined by the base font of the browser. This base font is typically Times New Roman in 16 pixels.

When you use the heading elements, the most important heading on a page should be an h1 element. You should only code one h1 element on each page. And you should only go down one level at a time, not jump down two or more levels to indicate less importance. In other words, the first heading level after an h1 should be an h2, not an h3. This structure helps search engines index your site, and it makes your pages more accessible to devices like screen readers.

Incidentally, HTML provides for three more levels below h3 with tags that range from h4 through h6. However, you shouldn't need these levels for the normal structure of a web page.

An attribute that should be coded in the html element for each page

Attribute	Description
`lang`	Identifies the language for the page

The block elements for headings and paragraphs

Element	Description
`h1`	A level-1 heading with content in bold at 200% of the base font size
`h2`	A level-2 heading with content in bold at 150% of the base font size
`h3`	A level-3 heading with content in bold at 117% of the base font size
`p`	A paragraph of text at 100% of the base font size

HTML that uses these block elements

```
<html lang="en">
    <h1>San Joaquin Valley Town Hall Programs</h1>
    <h2>Pre-lecture coffee at the Saroyan</h2>
    <p>Join us for a complimentary coffee hour, 9:15 to 10:15 a.m. on the
       day of each lecture. The speakers usually attend this very special
       event.</p>
    <h2>Post-lecture luncheon at the Saroyan</h2>
    <p>Extend the excitement of Town Hall by purchasing tickets to the
       luncheons.</p>
</html>
```

The block elements in a web browser

SEO guidelines

- Use the h1 tag to identify the most important information on the page, and only code one h1 tag on each page. Then, decrease one level at a time to show lower levels of importance.

Description

- *Block elements* are the building blocks of a website. They always start on a new line and take up the full space of the area that they're in.
- The base font size and the spacing above and below headings and paragraphs are determined by the browser, but you can change those values by using CSS.

Figure 3-2 How to code the lang attribute, headings, and paragraphs

How to code the structural elements

Figure 3-3 presents the structural elements that were introduced with HTML5. To illustrate, the example in this figure shows how the header, main, and footer elements can be used to divide a web page into three parts. Later, this structure makes it easy to use CSS to change the page layout and format the content.

All of these structural elements are block elements. But unlike the heading and paragraph elements, you can nest other block elements within these structural elements. In the HTML code in this figure, for example, you can see that an h1 element is nested within the header element, and a <p> element is nested within both the main and footer elements.

Although this figure only illustrates the use of three of the seven elements in the table, you'll see the nav element used in the web page at the end of this chapter. And in later chapters, you'll see how the aside element is used for a sidebar within the main element; how the section element is used for other content within the main element; and how the article element is used for an article about a speaker.

But at this point, please note that all the HTML elements that you've learned about so far in this chapter have indicated the type of content they contain. For example, an h1 element identifies the top-level heading for a page, and the main element identifies the main content for a page. Elements like these are called *semantic elements* because they give meaning to their content, and the use of these elements can be referred to as *HTML semantics*.

Semantics of course are good for both the developer and the browser. And that in turn means that they improve both web accessibility and search engine optimization.

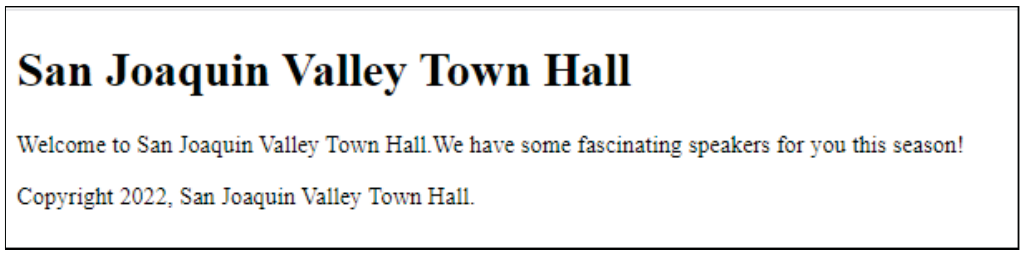
The primary HTML structural elements

Element	Contents
`header`	The header for a page
`main`	The main content for a page
`section`	A generic section of a document that doesn't indicate the type of content
`article`	A composition like an article in the paper
`nav`	A section of a page that contains links to other pages or placeholders
`aside`	A section of a page like a sidebar that is related to the content that's near it
`footer`	The footer for a page

A page that's structured with header, main, and footer elements

```
<html lang="en">
<body>
    <header>
        <h1>San Joaquin Valley Town Hall</h1>
    </header>
    <main>
        <p>Welcome to San Joaquin Valley Town Hall. We have some
           fascinating speakers for you this season!</p>
    </main>
    <footer>
        <p>Copyright 2022 San Joaquin Valley Town Hall.</p>
    </footer>
</body>
</html>
```

The page displayed in a web browser



Accessibility and SEO guideline

- Use the HTML structural elements to indicate the structure of your pages.

Description

- All of the HTML structural elements are block elements that can contain other block elements.
- Since structural elements like these identify the contents of the elements, they are called *semantic elements* and the process of using them is called *HTML semantics*.

Figure 3-3 How to code the structural elements

When and how to use div elements

But what happens if you want to divide one of the structural elements into two or more parts and there isn't a semantic element for that? Then, you can use div elements as shown in figure 3-4. Here, the first example shows how two div elements can be used to divide a section element into two divisions. Then, you can use CSS to lay out and format these divisions separately.

The second example in this figure shows how div elements were used before HTML5 introduced the structural elements. Here, you can see that the div elements divide the page into header, main, and footer divisions by using id attributes to identify them. But the div elements aren't semantic and don't help web accessibility or search engine optimization. That's why you don't want to use div elements when you can use the structural elements.

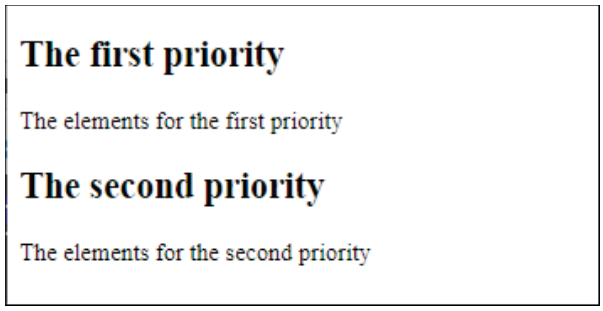
The div element

Element	Description
div	A block element that can be used to divide a structural element into divisions

How to use div elements to structure the content within a section

```
<section>
    <div>
        <h1>The first priority</h1>
        <p>The elements for the first priority</p>
    </div>
    <div>
        <h1>The second priority</h1>
        <p>The elements for the second priority</p>
    </div>
</section>
```

The resulting web page



How div elements were used before HTML5

```
<body>
    <div id="header">
        <h1>San Joaquin Valley Town Hall</h1>
    </div>
    <div id="main">
        <p>Welcome to San Joaquin Valley Town Hall. We have some
           fascinating speakers for you this season!</p>
    </div>
    <div id="footer">
        <p>Copyright 2015, San Joaquin Valley Town Hall.</p>
    </div>
</body>
```

Accessibility and SEO guidelines

- Only use div tags when the HTML semantic elements don't apply.

Description

- Before HTML5, div elements were used to provide the structure for a web page. But now, div elements should only be used to provide structure within a semantic element.

Figure 3-4 When and how to use div elements

Other elements for presenting text

The next two figures show how to use inline elements, other block elements, and character entities to present the text for a web page.

How to code the inline elements for text

In contrast to a block element, an *inline element* doesn't start on a new line. Instead, an inline element is coded within a block element. In figure, 3-5, you can see how 14 inline elements can be used to provide and format the text for a web page.

The first table in this figure presents elements that you can use to identify inline content. For instance, you can use the abbr element to identify an abbreviation, the <q> element to identify a quotation, and the cite element to identify the source of a block element like a quotation.

The second table presents elements that you can use to format content. For instance, the em and strong elements can be used to provide two different levels of emphasis. And the sub and sup elements let you provide for subscripts and superscripts.

The examples illustrate a few of these elements. Note that the <q> element adds quotation marks to a quotation. The em element converts the text to italics. And the strong element converts the text to boldface.

For the record, you can also use the <i> and <b> tags to italicize and boldface text. But since they don't imply any special meaning, it's better to use the elements in the second table.

Now notice the last element in the second table. It is the span element, which is an inline element with no meaning. Although you will rarely need to use it because it's better to use elements with meaning, you will see it used by the JavaScript code in chapter 18. You can also use it to apply CSS formatting to a portion of text in a block element when that formatting doesn't fall into one of the other categories in the second table.

Inline elements that identify the content

Element	Use
`abbr`	An abbreviation
`cite`	A bibliographic citation like a book title
`code`	Computer code that's displayed in a monospaced font
`dfn`	A term that is defined elsewhere
`kbd`	A keyboard entry that is displayed in a monospaced font
`q`	A quotation that is displayed within quotation marks
`samp`	A sequence of characters (a sample) that has no other meaning
`time`	A date or a date and time in a standard format
`var`	A computer variable that is displayed in a monospaced font

Inline elements that format text

Element	Use
`em`	To indicate that the content should be emphasized with an italic font
`small`	To display "fine print" such as footnotes in a smaller font
`strong`	To indicate that the content should be strongly emphasized in a bold font
`sub`	To display the content as a subscript
`sup`	To display the content as a superscript
`span`	An inline element with no meaning that formatting can be applied to

HTML that uses some of the inline elements

```
<p>When the dialog box is displayed, enter <kbd>brock21</kbd>.</p>
<p>To quote Hamlet: <q>Conscience does make cowards of us all.</q></p>
<p>If you don't get 78% or more on your final, <em>you won't pass.</em></p>
<p>Save a bundle at our <strong>big yearend sale</strong>.</p>
<p>The chemical symbol for water is H<sub>2</sub>O.</p>
```

The inline elements in a web browser

When the dialog box is displayed, enter `brock21`.

To quote Hamlet: "Conscience does make cowards of us all."

If you don't get 78% or more on your final, *you won't pass*.

Save a bundle at our **big yearend sale**.

The chemical symbol for water is H_2O.

Description

- An *inline element* is coded within a block element and doesn't begin on a new line.
- Although you can use the <b> and <i> elements to apply bold and italics to text, it's better to use the formatting elements in this figure.

Figure 3-5 How to code the inline elements for text

How to use character entities and three block elements for text

Figure 3-6 shows how to use *character entities* and three more block elements for text. In the first table, you can see 12 of the many character entities that provide a way to include characters like ampersands (&) and degree symbols in your text. As you can see, all character entities start with an ampersand (&) and end with a semicolon (;). Then, the rest of the entity identifies the character it represents.

In the second table, you can see three more block elements for coding special types of text. For instance, the blockquote element can be used to display a quotation, and the address element can be used to present contact information. Like the structural elements, these elements are good semantically because they identify the contents.

The examples that follow illustrate how character entities and these block elements work. Here, the blockquote element is used for a quotation, but note that quotation marks aren't added to it. Then, the address element is used for contact information.

Last, two character entities are used within a <p> element. The first one provides the copyright symbol. The second one provides an ampersand.

This shows that you can't just type an ampersand in your text because the ampersand is the first character in a character entity. Similarly, because the left bracket (<) and right bracket (>) are used to identify HTML tags, you can't use those characters to represent less-than and greater-than signs. Instead, you need to use the < and > entities.

You should know, however, that in some cases the &, <, and > characters will work without coding them as character entities. If you want them to work for all purposes and pass all validation tests, though, it's best to code them as character entities.

Common HTML character entities

Entity	Character	Entity	Character
&amp;	&	&deg;	°
&lt;	<	&plusmn;	±
&gt;	>	&cent;	¢
&copy;	©	&lsquo;	‘ (opening single quote)
&reg;	®	&rsquo;	’ (closing single quote or apostrophe)
&trade;	™		non-breaking space

Block elements for special types of text

Element	Used for
blockquote	A quotation
address	Contact information for the developer or owner of a website
pre	Preformatted text with preserved whitespace and a monospaced font

HTML that uses these block elements and character entities

```
<p>Ernest Hemingway wrote:</p>
<blockquote>
    Cowardice, as distinguished from panic, is almost always simply
    a lack of ability to suspend the functioning of the imagination.
</blockquote>

<address>1-800-221-5528<br>
    <a href="emailto:murachbooks@murach.com">murachbooks@murach.com</a>
</address>

<p>&amp;copy; 2022 Mike Murach &amp;amp; Associates, Inc.</p>
```

The examples in a web browser

Ernest Hemingway wrote:

> Cowardice, as distinguished from panic, is almost always simply a lack of ability to suspend the functioning of the imagination.

1-800-221-5528
murachbooks@murach.com

© 2022 Mike Murach & Associates, Inc.

Description

- *Character entities* are used to display special characters in an HTML document.
- These block elements are good semantically because they identify the type of content.

Figure 3-6 How to use character entities and three block elements for text

How to code links, lists, and images

Because you'll use links, lists, and images in most of the web pages that you develop, the topics that follow introduce you to these elements. But first, you need to know how to code absolute and relative URLs so you can use them with your links and images.

How to code URLs

Figure 3-7 presents some examples of absolute and relative URLs. To help you understand how these examples work, the diagram at the top of this figure shows the folder structure for the website used in the examples. As you can see, the folders are organized into three levels. The root folder for the site contains five subfolders, including the folders that contain the images and styles for the site. Then, the books folder contains subfolders of its own.

In chapter 1, you learned the basic components of an *absolute URL*, which includes the domain name of the website. This is illustrated by the first group of examples in this figure. Here, both URLs refer to pages at www.murach.com. The first URL points to the index.html file in the root folder of this website, and the second URL points to the toc.html file in the root/books/php folder.

When used within the code for the web pages of a site, an absolute URL is used to refer to a file in another website. By contrast, a *relative URL* is used to refer to a file within the same website. And as this figure shows, there are two types of relative URLs.

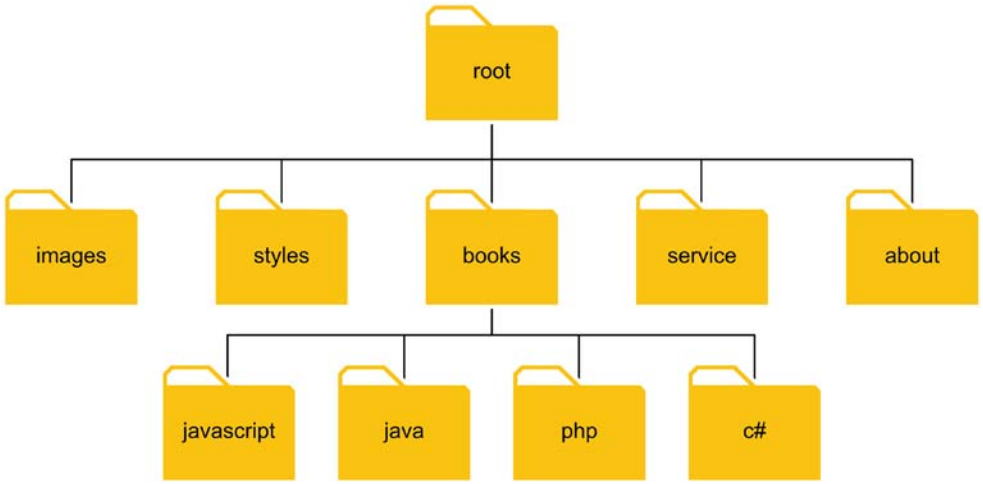
In a *root-relative path*, the path is relative to the root folder of the website. This is illustrated by the second group of examples. Here, the leading slash indicates the root folder for the site. As a result, the first path refers to the login.html file in the root folder, and the second path refers to the logo.gif file in the images folder.

In a *document-relative path*, the path is relative to the current document. This is illustrated by the third group of examples. Here, the assumption is that the paths are coded in a file that is in the root folder for a website. Then, the first path refers to a file in the images subfolder of the root folder, and the second path refers to a file in the php subfolder of the books subfolder. This illustrates paths that navigate down the levels of the folder structure.

But you can also navigate up the levels with a document-relative path. This is illustrated by the fourth group of examples. Here, the assumption is that the current document is in the root/books folder. Then, the first path goes up one level for the index.html file in the root folder. The second path also goes up one level to the root folder and then down one level for the logo.gif file in the images folder.

This shows that there's more than one way to code the path for a file. If, for example, you're coding an HTML file in the root/books folder, you can use either a root-relative path or a document-relative path to get to the images subfolder. If this is confusing right now, you'll get used to it when you start coding your own pages.

A simple website folder structure



Examples of absolute and relative URLs

Absolute URLs

```
http://www.murach.com/index.html
http://www.murach.com/books/php/toc.html
```

Root-relative paths

```
/login.html                (refers to root/login.html)
/images/logo.gif           (refers to root/images/logo.gif)
```

Document-relative paths that navigate down from the root folder

```
images/logo.gif            (refers to root/images/logo.gif)
books/php/overview.html    (refers to root/books/php/overview.html)
```

Document-relative paths that navigate up from the root/books folder

```
../index.html              (refers to root/index.html)
../images/logo.gif         (refers to root/images/logo.gif)
```

Description

- When you code an *absolute URL*, you code the complete URL including the domain name for the site. Absolute URLs let you display pages at other websites.
- When you code a *relative URL*, you base it on the current folder, which is the folder that contains the current page.
- A *root-relative path* is relative to the root folder of the website. It always starts with a slash. Then, to go down one subfolder, you code the subfolder name and a slash. To go down two subfolders, you code a second subfolder name and another slash. And so on.
- A *document-relative path* is relative to the folder the current document is in. Then, to go down one subfolder, you code the subfolder name followed by a slash. To go down two subfolders, you code a second subfolder name followed by another slash. And so on.
- You can also go up in a document-relative path. To go up one level from the current folder, you code two periods and a slash. To go up two levels, you code two periods and a slash followed by two more periods and a slash. And so on.

Figure 3-7 How to code URLs

How to code links

Most web pages contain *links* that go to other web pages or web resources. To code a link, you use the <a> element (or anchor element) as shown in figure 3-8. Because this element is an inline element, you usually code it within a block element like a <p> element.

In most cases, you'll code only the href attribute for the <a> element. This attribute specifies the URL for the resource you want to link to. The examples in this figure illustrate how this works.

The first example uses a relative URL to link to a page in the same folder as the current page. The second example uses a relative URL to link to a page in a subfolder of the parent folder. The third example uses a relative URL to link to a page based on the root folder. And the last example uses an absolute URL to link to a page at another website.

By default, links are underlined when they're displayed in a browser to indicate that they're clickable. As a result, most web users have been conditioned to associate underlined text with links. Because of that, you should avoid underlining any other text.

When a link is displayed, it has a default color depending on its state. For instance, a link that hasn't been visited is displayed in blue, and a link that has been visited is displayed in purple. Note, however, that you can use CSS as described in the next chapter to change these settings.

When you create a link that contains text, the text should clearly indicate the function of the link. For example, you shouldn't use text like "click here" because it doesn't indicate what the link does. Instead, you should use text like that in the examples in this figure. In short, if you can't tell where a link goes by reading its text, you should rewrite the text. This improves the accessibility of your site, and it helps search engines index your site.

The primary attribute of the <a> element

Attribute	Description
`href`	Specifies a relative or absolute URL for a link.

A link to a web page in the same folder

```
<p>Go view our <a href="products.html">product list</a>.</p>
```

A link to a web page in a subfolder of the parent folder

```
<p>Read about the <a href="../company/services.html">services we
provide</a>.</p>
```

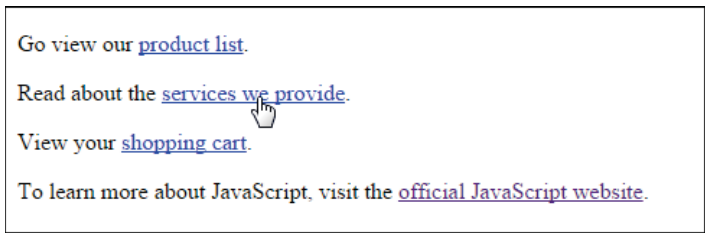
A link to a web page based on the root folder

```
<p>View your <a href="/orders/cart.html">shopping cart</a>.</p>
```

A link to a web page at another website

```
<p>To learn more about JavaScript, visit the
<a href="http://www.javascript.com/">official JavaScript web site</a>.</p>
```

The links in a web browser



SEO and accessibility guideline

- The content of a link should be text that clearly indicates where the link is going.

Description

- The <a> element is an inline element that creates a *link* that loads another web page. The href attribute of this element identifies the page to be loaded.
- The text content of a link is underlined by default to indicate that it's clickable.
- If a link hasn't been visited, it's displayed in blue. If it has been visited, it's displayed in purple. But you can change these values using CSS.
- If the mouse hovers over a link, the cursor is changed to a hand with the finger pointed as shown above.
- For more information on coding <a> elements, see chapter 7.

Figure 3-8 How to code links

How to code lists

Figure 3-9 shows how to code the two basic types of lists: ordered lists and unordered lists. To create an *unordered list*, you use the ul element. Then, within this element, you code one li (list item) element for each item in the list. The content of each li element is the text that's displayed in the list. By default, when a list is displayed in a browser, each item in an unordered list is preceded by a bullet. However, you can change that bullet with CSS.

To create an *ordered list*, you use the ol element, along with one li element for each item in the list. This works like the ul element, except that the items are preceded by numbers rather than bullets when they're displayed in a browser. In this case, you can change the type of numbers that are used with CSS.

The two lists shown in this figure illustrate how this works. Here, the first list displays the names of several programming languages, so these items don't need to reflect any order. By contrast, the second list identifies three steps for completing an order. Because these steps must be completed in a prescribed sequence, they're displayed in an ordered list.

When you work with the li element, you should be aware that it can contain text, inline elements, or block elements. For example, an li element can contain an <a> element that defines a link. In fact, it's a best practice to code a series of links within an unordered list. You'll see an example of that later in this chapter, and you'll learn all about it in chapter 7.

Elements that create ordered and unordered lists

Element	Description
`<ul>`	Creates an unordered list.
`<ol>`	Creates an ordered list.
`<li>`	Creates a list item for an unordered or ordered list.

HTML that creates two lists

```
<p>We have books on a variety of languages, including</p>
<ul>
    <li>JavaScript</li>
    <li>PHP</li>
    <li>Java</li>
    <li>C#</li>
</ul>

<p>You will need to complete the following steps:</p>
<ol>
    <li>Enter your billing information.</li>
    <li>Enter your shipping information.</li>
    <li>Confirm your order.</li>
</ol>
```

The lists in a web browser

We have books on a variety of languages, including

- JavaScript
- PHP
- Java
- C#

You will need to complete the following steps:

1. Enter your billing information.
2. Enter your shipping information.
3. Confirm your order.

Description

- The two basic types of lists are *unordered lists* and *ordered lists*.
- By default, an unordered list is displayed as a bulleted list, and an ordered list is displayed as a numbered list.
- For more information on coding lists, see chapter 7.

Figure 3-9 How to code lists

How to include images

Images are an important part of most web pages. To display an image, you use the img element shown in figure 3-10. This is an inline element that's coded as an empty tag. In the example in this figure, this tag is coded before an h1 element. The h1 element is displayed below it, though, because it's a block element that starts on a new line.

The src (source) attribute of an img element specifies the URL of the image that you want to display, and it is required. For instance, the src attribute for the image in this example indicates that the image named murachlogo.gif can be found in the images subfolder of the current folder.

The alt attribute should also be coded for img elements. You typically use this attribute to provide information about the image in case it can't be displayed or the page is being accessed by a screen reader. This is essential for visually-impaired users.

In this example, the image is our company's logo, so the value of the alt attribute is set to "Murach Logo". If an image doesn't provide any meaning, however, you should code the value of the alt attribute as an empty string (""). You should do that, for example, when an image is only used for decoration.

You can use the height and width attributes of an img element to tell the browser what the size of an image is. That can help the browser lay out the page as the image is being loaded. Although you can also use the height and width attributes to render an image larger (known as "stretching") or smaller than the original image, it's better to use your image editor to make the image the right size. You'll learn more about working with images in chapter 11.

Attributes of the <img> element

Attribute	Description
`src`	The relative URL of the image to display. It is a required attribute.
`alt`	Alternate text that's displayed in place of the image. This text is read aloud by screen readers for users with disabilities. It is required.
`height`	The height of the image in pixels.
`width`	The width of the image in pixels.

An img element

```
<img src="images/murachlogo.gif" alt="Murach Logo" height="75">
<h1>Mike Murach &amp;amp; Associates, Inc.</h1>
```

The image in a web browser

Common image formats

- JPEG (Joint Photographic Experts Group)
- GIF (Graphic Interchange Format)
- PNG (Portable Network Graphics)

Accessibility guidelines

- For images with useful content, always code an alt attribute that describes the image.
- For images that are used for decoration, code the alt attribute with no value (`""`).

Description

- The img element is an inline element that is used to display an image that's identified by the src attribute.
- The height and width attributes can be used to indicate the size of an image so the browser can allocate the correct amount of space on the page. These attributes can also be used to size an image, but it's usually better to use an image editor to do that.
- JPEG files have the JPG extension and are used for photographs. GIF files are used for small illustrations and logos. And PNG files combine aspects of JPEG and GIF files.
- For more information on coding img elements, see chapter 11.

Figure 3-10 How to include images

A structured web page

Now that you've seen the HTML elements for structuring a web page, figure 3-11 presents a simple web page that uses these elements.

The page layout

The purpose of the web page in this figure is to provide information about a series of lectures being presented by a non-profit organization. It shows the default formatting for the HTML that's used for this page. In the next chapter, though, you'll learn how to use CSS to improve the formatting of this page.

The HTML file

The HTML for this web page starts with the DOCTYPE declaration, the html element, and head element. They illustrate the way these items should be coded for every web page that you create.

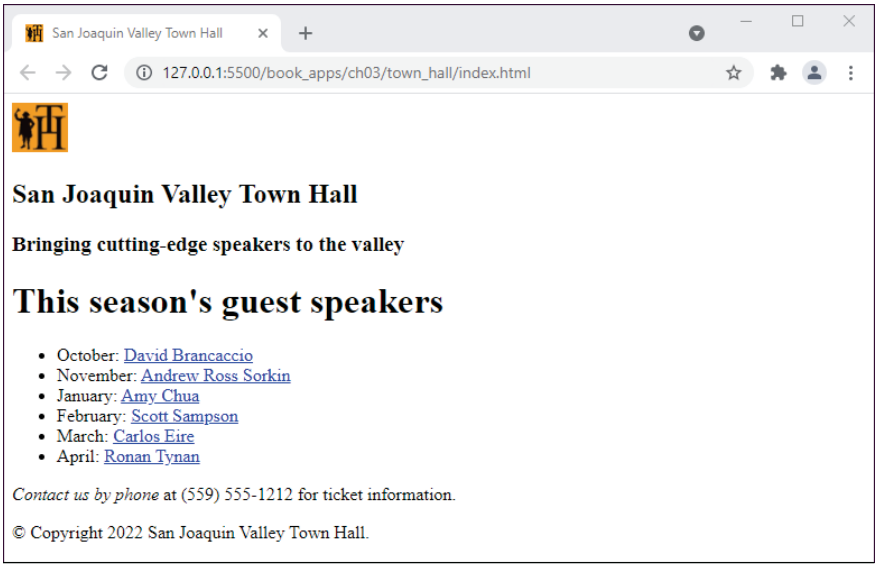
In the html element, you can see the use of the lang attribute. In the head element, you can see the coding for the charset meta element, the title element, and a favicon. Although it isn't' shown here, you should also include a meta element for the description in most web pages.

In the body element, you can see the use of the structural elements. Here, the header element contains one h2 element. The main element contains one h1 element, a nav element, and a <p> element. And the footer element contains one <p> element.

Within the nav element is an unordered list that contains five li elements (although the HTML is shown for only two of them). Then, each li element contains an <a> element. This is a best practice because a nav element should contain a series of links. It is also a best practice to code the <a> elements within an unordered list.

You can also see the use of an element that italicizes the first four words in the <p> element within the main element and a character entity that inserts the copyright symbol into the <p> element within the footer. But what's most important in this example is the use of the HTML structural elements and HTML semantics. That's good for the developer as well as for search engine optimization.

A web page that uses some of the HTML in this chapter



The HTML file for the web page

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>San Joaquin Valley Town Hall</title>
        <link rel="shortcut icon" href="images/favicon.ico">
    </head>
    <body>
        <header>
            <img src="images/logo.jpg" alt="Town Hall Logo" width="50">
            <h2>San Joaquin Valley Town Hall</h2>
        </header>
        <main>
            <h1>This season's guest speakers</h1>
            <nav>
                <ul>
                    <li>October: <a href="speakers/brancaccio.html">
                        David Brancaccio</a></li>
                    <li>November: <a href="speakers/sorkin.html">
                        Andrew Ross Sorkin</a></li>
                    ...
                </ul>
            </nav>
            <p><em>Contact us by phone</em> at (559) 555-1212 for ticket
               information.</p>
        </main>
        <footer>
            <p>&copy; Copyright 2022 San Joaquin Valley Town Hall.</p>
        </footer>
    </body>
</html>
```


Figure 3-11 A structured web page

Perspective

This chapter has presented most of the HTML elements that you will need as you develop web pages. That includes both block and inline elements. With those skills, you can create web pages that have the default formatting of the browser. Then, in the next three chapters, you will learn how to use CSS to format your pages so they look just the way you want them.

Terms

metadata
favicon
focus keywords
block element
semantic elements
HTML semantics
inline element
character entity
absolute URL
relative URL
root-relative path
document-relative path
link
ordered list
unordered list

Summary

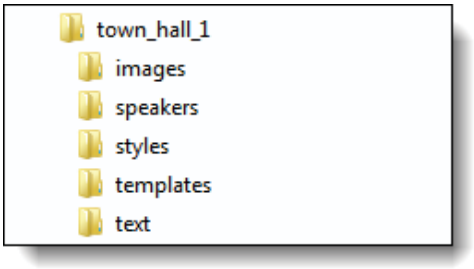
- The meta elements in the head section of a document provide *metadata* that includes the character set and description for the page. This can improve search engine optimization.
- The title in the head section of an HTML document provides the text that's displayed in the browser's tab for the page. This can also improve search engine optimization.
- You can code a link element in the head section to identify a custom icon called a *favicon* that appears in the browser's tab for the page. This icon may also appear in the browser's address bar or as part of a bookmark.
- *Block elements* are the primary content elements of a website, and each block element starts on a new line when it is rendered by a browser. Headings and paragraphs are common block elements.
- The HTML structural elements are block elements that let you structure a page into components like a header, a main portion, a navigation area, a sidebar, and a footer.
- Elements like header, main, footer, h1, and <p> are called *semantic elements* because they identify their contents. These elements are good for both developer and browser, and using them is referred as *HTML semantics*.
- Div elements can be used to provide structure within the structural elements. But they should only be used when there isn't an appropriate structural element.

- *Inline elements* are coded within block elements, and they don't start on new lines when they are rendered. HTML provides many inline elements that can be used to identify and format text.
- *Character entities* are used to display special characters like the ampersand and copyright symbol in an HTML document. In code, character entities start with an ampersand and end with a semicolon as in   (a non-breaking space).
- When you code an *absolute URL*, you code the complete URL including the domain name. When you code a *relative URL*, you can use a *root-relative path* to start the path from the root folder for the website or a *document-relative path* to start the path from the current document.
- The <a> element (or anchor element) is an inline element that creates a *link* that usually loads another page. By default, the text of an <a> element is underlined. Also, an unvisited link is displayed in blue and a visited link in purple.
- Lists are block elements that can be used to display both *unordered lists* and *ordered lists*. By default, these lists are indented with bullets before the items in an unordered list and numbers before the items in an ordered list.
- The img element is used to display an image file. The three common formats for images are *JPEG* (for photographs), *GIF* (for small illustrations and logos), and *PNG*, which combines aspects of JPEG and GIF.

About the exercises

In the exercises for chapters 3 through 8, you'll develop a new version of the Town Hall website. This version will be like the one in the text, but it will have different content, different formatting, and different page layouts.

As you develop this site, you will use this folder structure:



This is a realistic structure with images in the images folder, speaker HTML pages in the speakers folder, and CSS files in the styles folder. In addition, the text folder contains text files that will provide all of the content you need for your pages. That too is realistic because a web developer often works with text that has been written by someone else.

Exercise 3-1 Enter the HTML for the home page

In this exercise, you'll code the HTML for the home page. When you're through, the page should look like the one on the facing page, but with three speakers.

Open the starting page and get the contents for it

1. Use your text editor to open this HTML file:
   `\html_css_5\exercises\town_hall_1\c3_index.html`

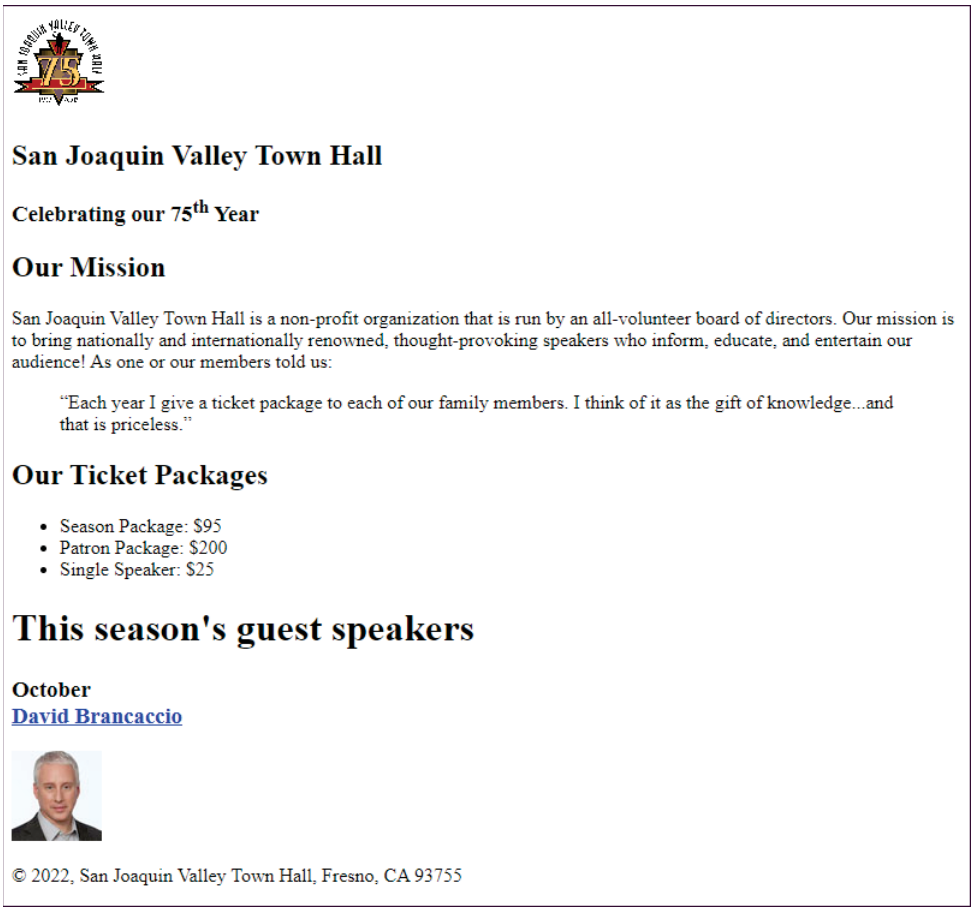
   Note that it contains the head section for this web page as well as a body section that contains a header element with some text, a main element, and a footer element with some text.
2. Use your text editor to open this text file:
   `\html_css_5\exercises\town_hall_1\text\c3_content.txt`

   Note that it includes all of the text for the main element.

Enter the header

3. Code the img element that gets the image at the top of the page from the images folder. To locate the image file, use this document-relative path: images/town_hall_logo.gif. Be sure to include the alt attribute, and set the height attribute of the image to 80.
4. Apply the h2 and h3 elements to the text in the header element. Then, test this page in Chrome. If necessary, correct the HTML and test again.

What the home page should look like



San Joaquin Valley Town Hall

Celebrating our 75th Year

Our Mission

San Joaquin Valley Town Hall is a non-profit organization that is run by an all-volunteer board of directors. Our mission is to bring nationally and internationally renowned, thought-provoking speakers who inform, educate, and entertain our audience! As one or our members told us:

> "Each year I give a ticket package to each of our family members. I think of it as the gift of knowledge...and that is priceless."

Our Ticket Packages

- Season Package: $95
- Patron Package: $200
- Single Speaker: $25

This season's guest speakers

October
David Brancaccio

© 2022, San Joaquin Valley Town Hall, Fresno, CA 93755

Enter the content for the main element

5. Copy all of the content for the main element from the txt file into the HTML file. Then, add an h1 tag to the heading "This season's guest speakers", and add h2 tags to these headings "Our Mission" and "Our Ticket Packages".
6. Add <p> tags to the first block of text after the "Our Mission" heading, and add blockquote tags to the second block of text as shown above.
7. Add the ul and li tags that are needed for the three items after the "Our Ticket Packages" heading. Then, test these changes and make any adjustments.
8. Format the name and month for the first speaker after the "This season's guest speakers" heading as one h3 element with a <br> in the middle that rolls the speaker's name over to a second line. Then, test and adjust. When that works, do the same for the next two speakers.
9. Enclose the name for each speaker in an <a> tag. The href attribute for each tag should refer to a file in the speakers subfolder that has the speaker's last name as the filename and html as the file extension.

10. After the h3 element for each speaker, code an img element that displays the image for the speaker, and be sure to include the alt attribute. The images are in the images subfolder, and the filename for each is the speaker's last name, followed by 75 (to indicate the image size), with jpg as the extension. Now, test and adjust.

Add character entities and the remaining formatting tags

11. Enclose the text in the footer in a <p> element. And use a character entity to add the copyright symbol to the start of the footer.
12. Add the sup tags that you need for raising the *th* in the second line of the header (as in 75th). Then, test these enhancements.

Test the links and add a link to one of the speaker pages

13. Click on the link for the first speaker page. This should display a page that gives the speaker's name and says "This page is under construction". If this doesn't work, fix the href attribute in the link and test again. To return to the first page, you can click the browser's Back button.
14. Open the sorkin.html file that's in the speakers subfolder. Then, add a link within a <p> element that says "Return to index page." To refer to the index.html file, you'll have to go up one level in the folder structure with a document-relative path like this: ../index.html. Now, test this link.
15. Test the index page in Chrome. If necessary, fix any problems and test again.