Copyright Information (bibliographic)

Document Type: Book Chapter

Title of Book: Murach's JavaScript and jQuery (4th edition)

Author(s) of Book: Mary Delamater, Zak Ruvalcaba

Chapter Title: Chapter 1 Introduction to Web Development

Author(s) of Chapter: Mary Delamater, Zak Ruvalcaba

Year: 2020

Publisher: Mike Murach & Associates, Inc.

Place of Publishing: the United States of America

1

Introduction to web development

This chapter presents the background concepts, terms, and skills that you need for developing JavaScript applications. That includes a quick review of the HTML and CSS skills that you need, and a tutorial on how to use Visual Studio Code, which is the source code editor that we used to develop the JavaScript apps for this book.

If you have some web development experience, you should be able to go through this chapter quickly by skimming the topics that you already know. But if you're new to web development, you should take the time to master the concepts and terms of this chapter.

How a web application works

A web application consists of many components that work together as they bring the application to your computer or mobile device. Before you can start developing JavaScript applications, you should have a basic understanding of how these components work together.

The components of a web application

The 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 *web browsers*. 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* is a large network that consists of many smaller networks. In a diagram like the one in this figure, the "cloud" represents the network or Internet that connects the clients and servers.

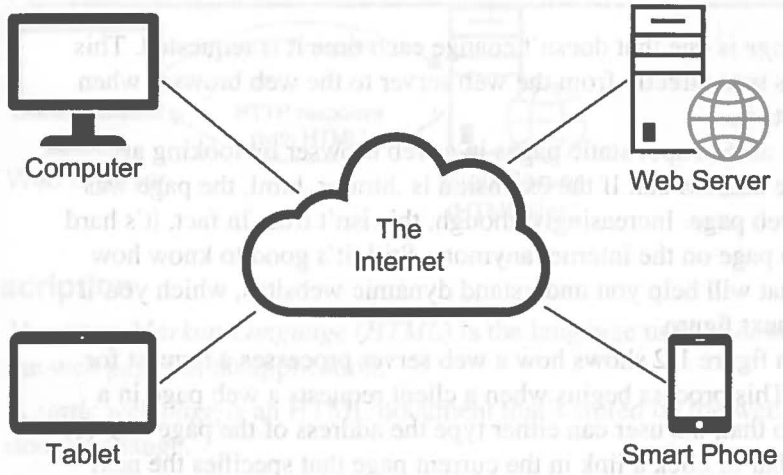
In general, you don't need to know how the cloud works. But you should have a general idea of what's going on.

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 in a LAN are typically in the same building or 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. 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 companies that need to be connected to the Internet.

The components of a web application



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. Today, the clients can be computers, smart phones like the iPhone, or tablets like the iPad.
- The *web server* returns the pages that are requested to the browser.
- A *network* connects the clients to the web server.
- An *intranet* is a *local area network* (or *LAN*) that connects computers that are near each other, usually within the same building.
- The *Internet* is a network that consists of many *wide area networks* (*WANs*), and each of those consists of two or more LANs. Today, the Internet is often referred to as "the Cloud", which implies that you really don't have to understand how it works.
- An *Internet service provider* (*ISP*) owns a WAN that is connected to the Internet.

Figure 1-1 The components of a web application

How static web pages are processed

A *static web page* is one that doesn't change each time it is requested. This type of web page is sent directly from the web server to the web browser when the browser requests it.

You used to be able to spot static pages in a web browser by looking at the extension in the address bar. If the extension is .htm or .html, the page was probably a static web page. Increasingly, though, this isn't true. In fact, it's hard to find a static web page on the internet anymore. Still, it's good to know how they work, since that will help you understand dynamic websites, which you'll learn about in the next figure.

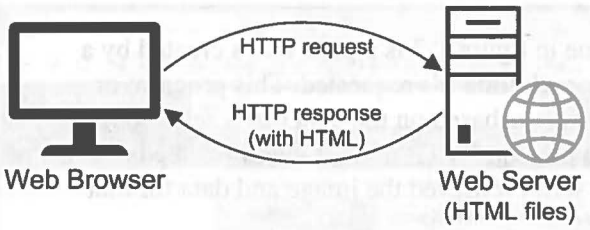
The diagram in figure 1-2 shows how a web server processes a request for a static web page. This process begins when a client requests a web page in a web browser. To do that, the user can either type the address of the page into the browser's address bar or click a link in the current page that specifies 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.

How a web server processes a static web page



Description

- *Hypertext Markup Language* (*HTML*) is the language used to define the content for the web pages of an application.
- A *static web page* is an HTML document that's stored on the web server and doesn't change.
- When the user requests a static web page, the 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 or script on the web server each time it's requested. This program or script is executed by an *application server* based on the data that's sent along with the HTTP request. In this example, the HTTP request identified the book that's shown. Then, the program or script retrieved 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. The 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 into the browser's address bar, click a link that specifies the dynamic page to load, or click a button that submits a form containing the data 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.

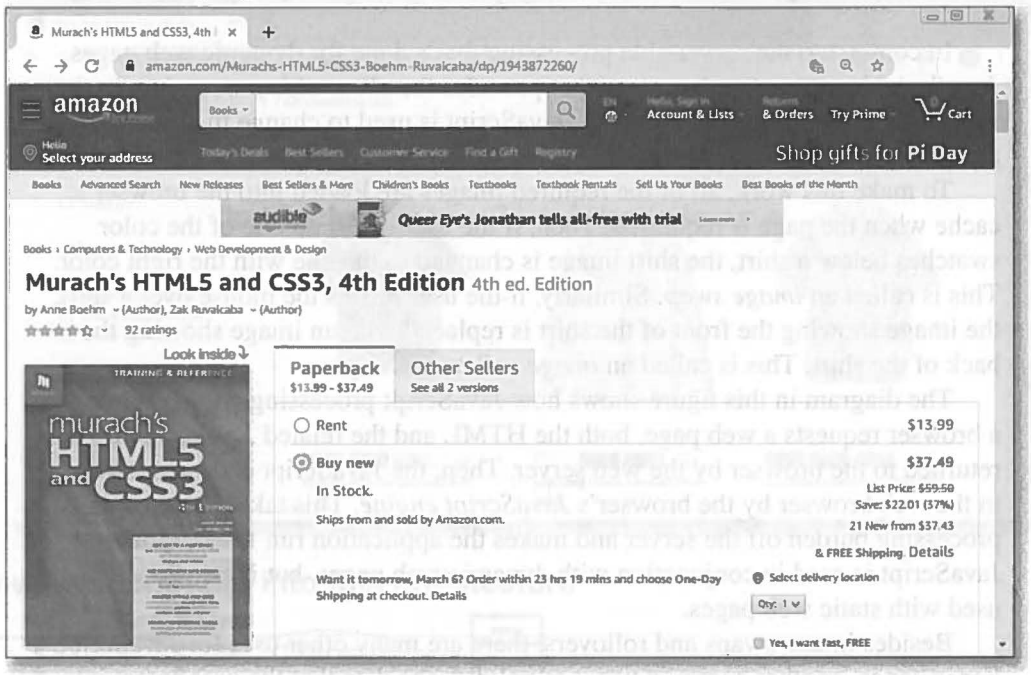
Next, the application server retrieves the appropriate program or script from the hard drive. It also loads any form data that the user submitted. Then, it executes the script. As the script executes, it generates the HTML for the web page. If necessary, the script will request data from a database server and use that data as part of the web page it is generating. The processing that's done on the application server can be referred to as *server-side processing*.

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.

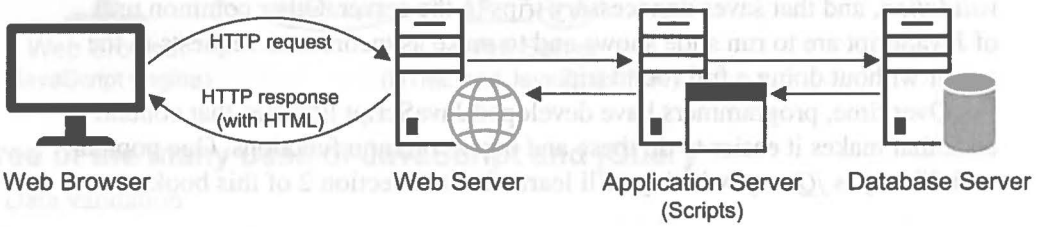
When the web browser receives the HTTP response, it renders the HTML and displays the web page. Note, however, that the web browser has no way to tell whether the HTML in the HTTP response is from a static page or a dynamic page. It just renders the HTML.

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*.

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 or script that is running on a server.
- 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 is used for client-side processing

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 web page in figure 1-4, for example, JavaScript is used to change the images that are shown without using server-side processing.

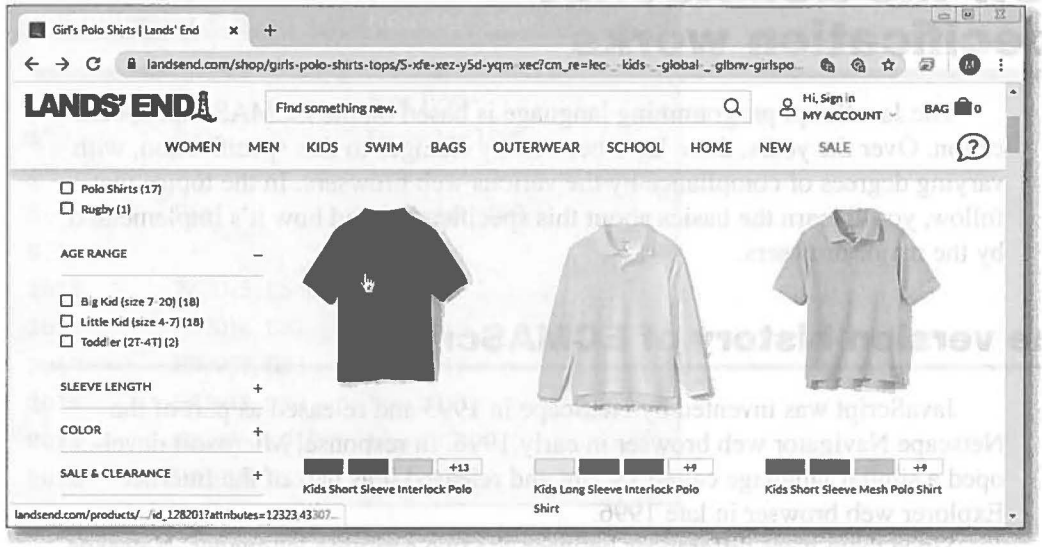
To make this work, all of the required images are loaded into the browser's cache when the page is requested. Then, if the user clicks on one of the color swatches below a shirt, the shirt image is changed to the one with the right color. This is called an *image swap*. Similarly, if the user moves the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

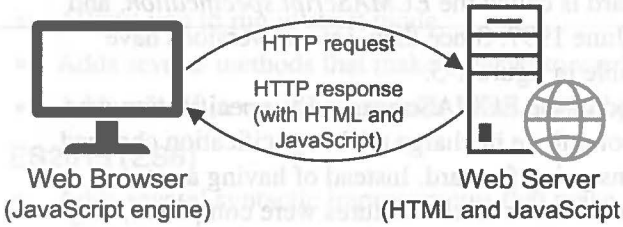
Besides image swaps and rollovers, there are many other uses for JavaScript. For instance, another common use is to validate the data that the user enters into an HTML form before it is sent to the server for processing. This is called *data validation*, and that saves unnecessary trips to the server. Other common uses of JavaScript are to run slide shows and to make asynchronous requests to the server without doing a full round trip.

Over time, programmers have developed JavaScript libraries that contain code that makes it easier to do these and other common functions. One popular such library is *jQuery*, which you'll learn about in section 2 of this book.

A web page with image swaps and rollovers



How JavaScript fits into this architecture



Three of the many uses of JavaScript and jQuery

- Data validation
- Image swaps and rollovers
- Slide shows

Description

- *JavaScript* is a *scripting language* that is run by the *JavaScript engine* of a web browser and controls the operation of the 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, its functions don't require a trip back to the server. This helps an application run more efficiently.
- *jQuery* is a JavaScript library that makes it easier to do many of the common functions that JavaScript is used for.

Figure 1-4 How JavaScript is used for client-side processing

How the ECMAScript specification works

The JavaScript programming language is based on the ECMAScript specification. Over the years, there have been many changes to this specification, with varying degrees of compliance by the various web browsers. In the topics that follow, you'll learn the basics about this specification and how it's implemented by the major browsers.

The version history of ECMAScript

JavaScript was invented by NetScape in 1995 and released as part of the Netscape Navigator web browser in early 1996. In response, Microsoft developed a similar language called JScript and released it as part of the Internet Explorer web browser in late 1996.

Since there were differences between the two scripting languages, Netscape gave JavaScript to the *European Computer Manufacturers Association (ECMA)* to develop a standard. The standard is called the *ECMAScript specification,* and the first version was released in June 1997. Since then, several versions have been released, as shown in the table in figure 1-5.

In June 2015 the sixth version of the ECMAScript, or ES, specification was released. At the same time, the committee in charge of the specification changed how it would release new versions going forward. Instead of having a set specification that they would release when all the features were completed, they moved to yearly releases of features that had been approved to that point. Thus, the version released in 2015 was officially named ECMAScript 2015, although you'll often see it referred to as ES6. After that, the versions are usually referred to by the year, such as ECMAScript 2016, or ES2016.

ES5 and ES6 represented significant updates to the ECMAScript specification, while ES2016 added only a handful of smaller features. Since then, ES2017, ES2018, ES2019, and ES2020 added several important features. Many of these features are listed in this figure. In this book, you'll learn how to use the most useful features of all the releases, including ES2020.

The versions and release dates of the ECMAScript specification

Version	AKA	Release date
1		June 1997
2		June 1998
3	ES3	December 1999
4		Abandoned (never released)
5	ES5	December 2009
5.1		June 2011
2015	ES2015, ES6	June 2015
2016	ES2016, ES7	June 2016
2017	ES2017, ES8	June 2017
2018	ES2018, ES9	June 2018
2019	ES2019, ES10	June 2019
2020	ES2020, ES11	June 2020

Some of the important additions in the recent versions

ES5

- Allows you to run in strict mode.
- Adds several methods that make it easier to work with arrays and objects.
- Adds a built-in way to work with JavaScript Object Notation (JSON).

ES2015 (ES6)

- Adds several syntactic improvements that make code easier to read and understand.
- Adds block scope and easier ways to work with classes.
- Adds arrow functions, iterators, and Promises for working with asynchronous code.

ES2016

- Adds a simpler syntax for computation with powers.
- Adds a method to check if an array includes a specified element.

ES2017

- Adds async functions and the await keyword for working with Promises.

ES2018

- Adds asynchronous iteration.
- Adds more regular expression features.

ES2019

- Adds new string, array, and object methods.
- Adds improvements to the JSON object.

ES2020

- Adds a new BigInt data type.
- Adds new operators for dealing with nulls.

Figure 1-5 The version history of ECMAScript

Browser support for ECMAScript

The table in figure 1-6 presents the versions of the major browsers that support the ECMAScript specification through ES2019. These browsers also support the major new features of ES2020. This means that these browsers support all the ECMAScript features that you'll learn about in this book

However, for earlier versions of these browsers, or for other browsers, the support for the later versions of the ECMAScript specification may be spottier. In particular, the Internet Explorer (IE) browser struggles to support the more recent versions of ECMAScript. Earlier versions of IE, such as IE6, were notoriously noncompliant with the specification. Later versions had better compliance, but still lagged behind the other browsers, and that continues to this day.

So, if you want your web pages to support web browsers that aren't included in this table, you'll need to make some adjustments. To start, you can go to the website at the URL in this figure to see which browsers support which features. Then, you can use only those features that are supported by the browsers you want to support.

The other approach is to add workarounds to your code that will make many of the ECMAScript features work in browsers that don't support them. You can also learn more about that by going to the website at the URL in this figure.

The browsers that support the ECMAScript specification

Browser	Version
Chrome	79 and above
Edge	79 and above
Firefox	68 and above
Opera	66 and above
Safari	12.1 and above

The URL for a browser compatibility table

`http://kangax.github.io/compat-table/`

Description

- At the time of this writing, the features of ECMAScript through ES2020 are supported by all the modern browsers listed here. Because of that, you can safely use the JavaScript presented in this book with these browsers.
- If you want to support other browsers, such as IE, or older versions of these browsers, you can use the website presented here to see which features are supported by which browsers.
- The website presented here also provides information about polyfill files and transpilers that you can use to allow browsers that don't fully support the ECMAScript specification to use your website.

Figure 1-6 Browser support for ECMAScript

The components of a JavaScript application

When you develop a JavaScript application, you use HTML to define the content and structure of the page. You use CSS to format that content. And you use JavaScript to do the client-side processing. This is illustrated by the Email List application that is presented in the next three figures.

Figure 1-7 starts with the user interface for the application. It asks the user to make three entries and then click on the Join List button. The asterisks to the right of the text boxes for the entries indicate that these entries are required.

When the user clicks on the button, JavaScript checks the entries to make sure they're valid. If they are, the entries are sent to the web server for server-side processing. If they aren't, messages are displayed so the user can correct the entries. This is a common use of JavaScript called *data validation* that saves a trip to the server when the entries are invalid.

You might have noticed that the user interface in this figure isn't much to look at. This is what a plain HTML document with no formatting looks like. In the next figure, though, you'll see how applying some CSS can improve its appearance.

The HTML

HyperText Markup Language (*HTML*) is used to define the content and structure of a web page. In figure 1-7, you can see the HTML for the Email List application. In general, this book assumes that you are already familiar with HTML, but here are a few highlights.

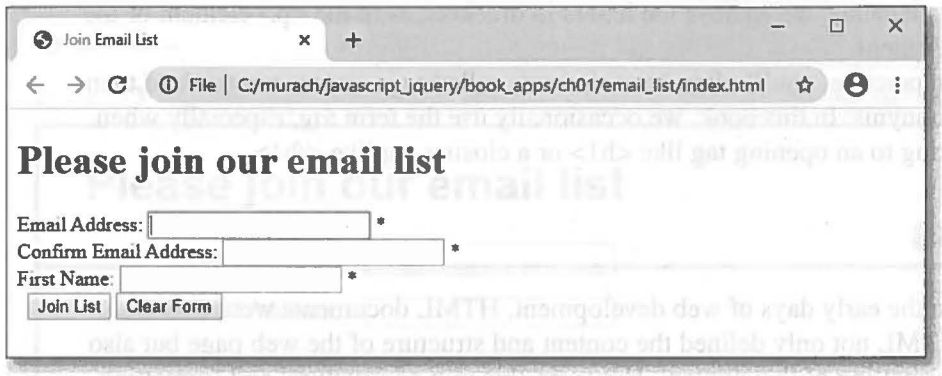
First, note that this document starts with a DOCTYPE declaration. This declaration is the one you'll use with HTML5, and you must code it exactly as it's shown here. In this book, all of the applications use HTML5.

Second, in the head section of the HTML document, you can see a meta element that specifies that UTF-8 is the character encoding that's used for the page. After that, there's another meta element that sets the viewport to help the page display properly on mobile devices. Then, there's a title element that contains the text that will display in the browser's tab. Finally, there's an HTML comment indicating that any link and style elements go here in the head element. You'll see an example of a link element for a CSS file in the next figure.

Third, in the body section, you can see the use of a main element, which is one of the HTML5 elements that we'll be using throughout this book. Within this element, you can see the use of h1, form, div, label, input, and span elements. The body section ends with an HTML comment indicating that any script elements go at the end of this section. Although script elements can also go in the head element, it's considered a best practice to put them at the end of the body section. In figure 1-9, you'll see an example of a script element for a JavaScript file.

In this book, as you've just seen, we refer to *HTML elements* like the <link>, <script>, <main>, and <h1> elements as the link, script, main, and h1 elements.

The HTML file in a browser before CSS is applied to it



The code for the HTML file named index.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Join Email List</title>
    <!-- link and style elements go here -->
</head>
<body>
    <main>
        <h1>Please join our email list</h1>
        <form id="email_form" action="join.html" method="get">
            <div>
                <label for="email_1">Email Address:</label>
                <input type="text" id="email_1" name="email_1">
                <span id="email_1_error">*</span>
            </div>
            <div>
                <label for="email_2">Re-enter Email Address:</label>
                <input type="text" id="email_2" name="email_2">
                <span id="email_2_error">*</span>
            </div>
            <div>
                <label for="first_name">First Name</label>
                <input type="text" id="first_name" name="first_name">
                <span id="first_name_error">*</span>
            </div>
            <div>
                <label> </label>
                <input type="submit" id="join_list" value="Join List">
                <input type="button" id="clear_form" value="Clear Form">
            </div>
        </form>
    </main>
    <!-- script elements go here -->
</body>
</html>
```

Description

- *HTML* (*HyperText Markup Language*) defines the structure and content of a web page.

Figure 1-7 The HTML for the web page

However, to prevent confusion when referring to one-letter elements like the p and a elements, we enclose the letters in brackets, as in the <p> element or the <a> element.

In practice, you'll often hear *elements* called *tags* so you can think of them as synonyms. In this book, we occasionally use the term *tag*, especially when referring to an opening tag like <h1> or a closing tag like </h1>.

The CSS

In the early days of web development, HTML documents were coded so the HTML not only defined the content and structure of the web page but also the formatting of that content. However, this mix of structural and formatting elements made it hard to edit, maintain, and reformat the web pages.

Today, *Cascading Style Sheets* (*CSS*) let you separate the formatting from the content and structure of a web page. As a result, the formatting that was once done with HTML should now be done with CSS.

In figure 1-8, then, you can see the link element that links the external CSS file to the HTML. As you saw in figure 1-7, this link element goes in the HTML head element. You can also see how this CSS has changed the appearance of the page in the browser.

After that, you can see the CSS that's used to format the HTML in the last figure. Here again, this book assumes that you are already familiar with CSS, but here is a quick description of what this CSS is doing.

In the style rule for the body element, the font-family property sets the font for the entire document, the background-color property sets the background color of the browser window, the margin property centers the body in the browser window, the width property sets the width of the body to 670 pixels, the border property puts a blue border around the body, and the padding property puts space between the contents and the right, left, and bottom borders. This is typical CSS for the applications in this book.

Similarly, the style rules for the h1, div, label, input, and span elements are intended to make these elements look better. For instance, the style rule for the h1 element sets the font color to blue, and the style rule for div elements sets the bottom margin so there's horizontal space after each div.

The style rule for the labels makes them display as block elements, so things like width can be applied to them, but also as inline elements, so they can be on the same line as other elements. Then, this style rule sets the width of the labels to 11ems, and it aligns the text for the labels on the right.

The style rule for the input elements sets the left margin so there's space between the labels and the text boxes. It also sets the right margin so there's space between the text boxes and span elements.

Last, the style rule for the span elements sets the text color to red. When the HTML page is first loaded, these span elements only contain asterisks (*) to indicate that these entries are required. But the JavaScript changes those asterisks to error messages if the related entries are invalid, and it removes the asterisks if the related entries are valid.

The web page in a browser after CSS has been applied to it

The link element in the HTML head element that applies the CSS file

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

The code for the CSS file named email_list.css

```
body {
    font-family: Arial, Helvetica, sans-serif;
    background-color: white;
    margin: 0 auto;
    width: 670px;
    border: 3px solid blue;
    padding: 0 2em 1em;
}
h1 {
    color: blue;
}
div {
    margin-bottom: 1em;
}
label {
    display: inline-block;
    width: 11em;
    text-align: right;
}
input {
    margin-left: 1em;
    margin-right: 0.5em;
}
span {
    color: red;
}
```

Description

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

Figure 1-8 The CSS for the web page

The JavaScript

Figure 1-9 shows how this application looks in a browser if the JavaScript finds any invalid data after the user clicks the Join List button. Here, you can see that error messages are displayed to the right of the user entries for the second and third text boxes. In other words, the JavaScript has actually changed the contents of the span elements.

When JavaScript changes the HTML for a page, it is called *DOM scripting*. That's because the JavaScript is actually changing the *Document Object Model* (or *DOM*) that's generated by the browser when the page is loaded. This DOM represents all of the elements and attributes that are coded in the HTML. Then, when JavaScript changes any aspect of the DOM, the change is immediately made to the browser display too.

After the browser display, this figure shows the script element that links the external JavaScript file to the HTML. This element is typically coded at the end of the body section, as shown in figure 1-7.

Then, this figure shows the JavaScript for this application. Since you are going to learn how all of this code works in the next six chapters, you may want to skip over this code right now. But if you have any programming experience, it may be worth taking a quick look at it. In that case, here are a few highlights.

To start, this code consists of four *functions* that consist of blocks of code. You'll learn more about how to code functions as you progress through this book. For now, just know that the $() function gets an element from the page, the joinList() function executes when the user clicks on the Join List button, the clearForm() function executes when the user clicks on the Clear Form button, and the anonymous function runs when the DOM has been loaded into the browser.

In the joinList() function, you can see four if-else statements that provide most of the logic for this application. Here, you can see that the if-else structures are similar to those in any modern programming language like Java, C#, or PHP. You can also see that declaring a variable or constant and assigning a value to it is done in a way that's similar to the way that it's done in other programming languages.

What's different about JavaScript is that it provides methods and properties that let you modify the DOM. For instance, the $() function uses the querySelector() method to get the first element in the document that matches the CSS selector that's passed to the function. Then, the first statement in the joinList() function uses the $() function to get the object that represents the first text box in the HTML. This statement also uses the value property to get the value that the user entered into that text box.

Later, the first if statement checks whether that value is an empty string (""), which means the user didn't make an entry. If it is, the JavaScript replaces the * in the span element for that text box with an error message. To do that, it uses this code:

```
$("#email_1_error").textContent = "Email is required.";
```

The web page in a browser with JavaScript used for data validation

The script element in the HTML body element that adds the JavaScript file

```
<script src="email_list.js"></script>
```

The code for the JavaScript file named email_list.js

```
const $ = selector => document.querySelector(selector);

const joinList = evt => {
    // get user entries from text boxes
    const email1 = $("#email_1").value;
    const email2 = $("#email_2").value;
    const firstName = $("#first_name").value;

    // check user entries
    let isValid = true;
    if (email1 == "") {
        $("#email_1_error").textContent = "Email is required.";
        isValid = false;
    } else {
        $("#email_1_error").textContent = "";
    }
    if (email1 != email2) {
        $("#email_2_error").textContent = "Emails must match.";
        isValid = false;
    } else {
        $("#email_2_error").textContent = "";
    }
    if (firstName == "") {
        $("#first_name_error").textContent = "First name is required.";
        isValid = false;
    } else {
        $("#first_name_error").textContent = "";
    }

    // cancel form submit if any user entries are invalid
    if ( !isValid ) {
        evt.preventDefault();
    }
};
```

Figure 1-9 The JavaScript for the web page (part 1)

The final if statement checks if any of the user's entries are invalid. If so, it uses the preventDefault() method to cancel the default method of the Join List submit button, which is to post the data in the form to the server. This prevents unnecessary round trips to the server.

In the clearForm() function, you can see three groups of code statements that modify the DOM. The first three statements clear any values in the text boxes, while the next three statements clear any error messages in the span elements. Finally, the last statement moves the browser's cursor to the first text box.

The last block of code is a call to the addEventListener() method, which attaches an anonymous function as the event handler for the DOMContentLoaded event. Then, within the anonymous function, the joinList() and clearForm() functions are attached as event handlers for the click events of the Join List and Clear Form buttons.

Although this code may look daunting right now, you'll see that it's all quite manageable. You'll also come to realize that DOM scripting is where JavaScript get its power.

The code for the JavaScript file named email_list.js (continued)

```
const clearForm = () => {
    // clear text boxes
    $("#email_1").value = "";
    $("#email_2").value = "";
    $("#first_name").value = "";

    // clear span elements
    $("#email_1_error").textContent = "*";
    $("#email_2_error").textContent = "*";
    $("#first_name_error").textContent = "*";

    // set focus on first text box after resetting the form
    $("#email_1").focus();
};

document.addEventListener("DOMContentLoaded", () => {
    // hook up click events for both buttons
    $("#join_list").addEventListener("click", joinList);
    $("#clear_form").addEventListener("click", clearForm);

    // set focus on first text box after the form loads
    $("#email_1").focus();
});
```

Figure 1-9 The JavaScript for the web page (part 2)

The HTML skills that you need for this book

Although this book assumes that you are already familiar with HTML, the next three topics present a quick review of the HTML skills that you're going to need for this book. If you don't already have these skills and you can't pick them up from the topics that follow, we recommend that you use *Murach's HTML5 and CSS3* as a reference while you're learning JavaScript.

How to use the HTML5 semantic elements

All of the applications in this book use the HTML5 semantic elements whenever they're appropriate. If you aren't already using them or at least familiar with them, figure 1-10 summarizes what you need to know.

In particular, the applications in this book may use the main, section, aside, and nav elements. That makes it easier to apply CSS to these elements because you don't have to code id attributes that are used by the CSS. Instead, you can apply the CSS to the elements themselves.

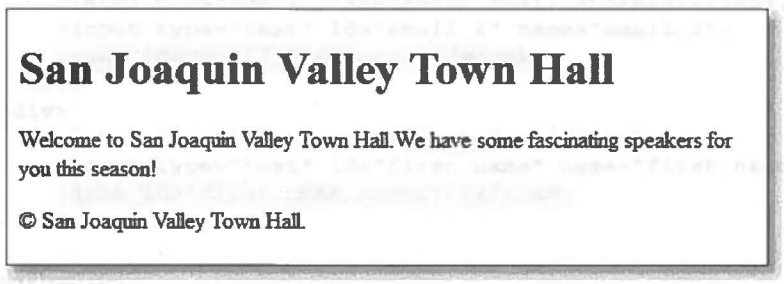
The primary HTML5 semantic elements

Element	Contents
`header`	The header for a page.
`main`	The main content of a page. Can only appear once per page, and cannot be the child of an article, aside, footer, header, or nav element.
`section`	A generic section of a document that doesn't indicate the type of content.
`article`	A composition like an article in the paper.
`aside`	A portion of a page like a sidebar that is related to the content that's near it.
`nav`	A portion of a page that contains links to other pages or placeholders.
`figure`	An image, table, or other component that's treated as a figure.
`footer`	The footer for a page.

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

```
<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>&copy; San Joaquin Valley Town Hall.</p>
    </footer>
</body>
```

The page displayed in a web browser



Description

- HTML5 provides *semantic elements* that you should use to structure the contents of a web page. Using these elements can be referred to as *HTML5 semantics*.
- All of the HTML5 elements are supported by the modern browsers.
- This book also uses standard HTML elements like h1 and h2 elements for headings, img elements for images, <a> elements for links, and <p> elements for paragraphs.

Figure 1-10 How to use the HTML5 semantic elements

How to use the div and span elements

If you've been using HTML for a while, you are certainly familiar with the div element. It has traditionally been used to divide an HTML document into divisions that are identified by id attributes. Then, CSS can use the ids to apply formatting to the divisions.

But now that HTML5 is available, div elements shouldn't be used to structure a document. Instead, they should only be used when the HTML5 semantic elements aren't appropriate.

Note, however, that div elements are often used in JavaScript applications. If, for example, a section element contains three h2 elements with each followed by a div element, JavaScript can be used to display or hide a div element whenever the heading that precedes it is clicked. This structure is illustrated by the first example in figure 1-11, and you'll see how this works in chapter 6.

Similarly, span elements have historically been used to identify portions of text that can be formatted by CSS. By today's standards, though, it's better to use elements that indicate the contents of the elements, like the cite, code, and <q> elements.

But here again, span elements are often used in JavaScript applications, as shown by the second example in this figure. In fact, you've just seen this in the Email List application. In that application, JavaScript puts the error messages in the appropriate span elements.

Notice that this code also uses div elements. Unlike in the first example, though, these elements are used so they can be formatted with CSS, and they're used because none of the HTML5 semantic elements are appropriate.

The div and span elements

Element	Description
div	A block element that provides a container for other elements.
span	An inline element that lets you identify text that can be formatted with CSS.

Div elements in the HTML for a JavaScript application

```
<section id="faqs">
    <h1>jQuery FAQs</h1>
    <h2>What is JavaScript?</h2>
    <div>
        // contents
    </div>
    <h2>What is jQuery?</h2>
    <div>
        // contents
    </div>
    <h2>Why is jQuery so popular?</h2>
    <div>
        // contents
    </div>
</section>
```

Span elements in the HTML for a JavaScript application

```
<div>
    <label for="email_1">Email Address:</label>
    <input type="text" id="email_1" name="email_1">
    <span id="email_1_error">*</span>
</div>
<div>
    <label for="email_2">Re-enter Email Address:</label>
    <input type="text" id="email_2" name="email_2">
    <span id="email_2_error">*</span>
</div>
<div>
    <label for="first_name">First Name</label>
    <input type="text" id="first_name" name="first_name">
    <span id="first_name_error">*</span>
</div>
```

Description

- Before HTML5, div elements were used to define the structure within the body of a document. The ids for these div elements were then used by the CSS to apply formatting to the elements.
- Today, the HTML5 semantic elements should be used to make the structure of a page more apparent. However, you will still use div elements to define blocks of code that are used in JavaScript applications.
- Before HTML5, span elements were often used to identify portions of text that you could apply formatting to.
- Today, a better practice is to use specific elements to identify content. However, you will still use span elements for some JavaScript applications.

Figure 1-11 How to use the div and span elements

How to use the basic HTML attributes

Figure 1-12 presents the HTML *attributes* that are commonly used in JavaScript applications. You should already be familiar with the *id attribute* that identifies one HTML element and with *class attributes* that can be applied to more than one HTML element. You should also be familiar with the *for attribute* that relates a label to an input element and with the *title attribute* that can be used to provide a tooltip for an element.

When you use JavaScript, you will commonly use the *name attribute* so the server-side code can access the data that is submitted to it. You will sometimes add or remove class attributes to change the formatting of elements. And you will sometimes use title attributes to provide text that's related to elements.

In practice, you usually use the same value for the id and name attributes of an element. For instance, the example in this figure uses "email" as the value of both the id and name attributes for the text box. That makes it easier to remember the attribute values.

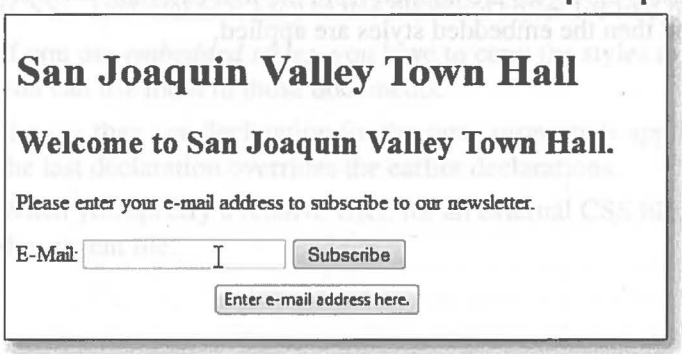
The basic HTML attributes

Attribute	Description
`id`	Specifies a unique identifier for an element that can be referred to by CSS.
`class`	Specifies one or more class names that can be referred to by CSS, and the same name can be used for more than one element. To code more than one class name, separate the class names with spaces.
`name`	Specifies a unique name for an element that is commonly used by the server-side code and can also be used by the JavaScript code.
`for`	In a label element, this attribute specifies the id of the control that it applies to.
`title`	Specifies additional information about an element. For some elements, the title appears in a tooltip when the user hovers the mouse over the element.

HTML that uses these attributes

```
<body>
    <h1>San Joaquin Valley Town Hall</h1>
    <h2 class="first_h2">Welcome to San Joaquin Valley Town Hall.</h2>
    <p>Please enter your e-mail address to subscribe to our
       newsletter.</p>
    <form id="email_form" name="email_form"
          action="join.html" method="get">
        <label for="email">E-Mail: </label>
        <input type="text" id="email" name="email"
               title="Enter e-mail address here.">
        <input type="button" value="Subscribe">
    </form>
</body>
```

The HTML in a web browser with a tooltip displayed for the text box



Description

- An *attribute* consists of an attribute name, an equals sign, and the value of the attribute enclosed in either single or double quotation marks.
- The *id* and *class attributes* are commonly used to apply CSS formatting. The id attribute is also commonly used by JavaScript code to access an element.
- The *name attribute* is commonly used by server-side code to access data.
- The *for attribute* in a label element is used to identify the control that it applies to.

Figure 1-12 How to use the basic HTML attributes

The CSS skills that you need for this book

Although this book assumes that you are already familiar with CSS, the next three topics present a quick review of the CSS skills that you're going to need for this book. If you don't already have these skills and you can't pick them up from the topics that follow, we recommend that you use *Murach's HTML5 and CSS3* as a reference while you're learning JavaScript.

How to provide the CSS styles for an HTML page

Figure 1-13 shows two ways that you can include CSS styles for an HTML document. First, you can code a link element in the head section of an HTML document that specifies a file that contains the CSS for the page. This is referred to as an *external style sheet*, and this is the method that's used for the applications in this book.

Second, you can code a style element in the head section that contains the CSS for the page. This can be referred to as *embedded styles*. In general, it's better to use external style sheets because that makes it easier to use them for more than one page. However, embedded styles can be easier to use for simple applications like the ones in this book because you don't need to create an extra file.

In some cases, you may want to use two or more external style sheets for a single page. You may even want to use both external style sheets and embedded styles for a page. In these cases, the styles are applied from the first external style sheet to the last one and then the embedded styles are applied.

Two ways to provide styles

Use an external style sheet by coding a link element in the head section

```
<link rel="stylesheet" href="styles/main.css">
```

Embed the styles in the head section

```
<style>
    body {
        font-family: Arial, Helvetica, sans-serif;
        font-size: 87.5%; }
    h1 { font-size: 250%; }
</style>
```

The sequence in which styles are applied

- Styles from an external style sheet
- Embedded styles

A head element that includes two external style sheets

```
<head>
    <title>San Joaquin Valley Town Hall</title>
    <link rel="stylesheet" href="../styles/main.css">
    <link rel="stylesheet" href="../styles/speaker.css">
</head>
```

The sequence in which styles are applied

- From the first external style sheet to the last

Description

- When you use *external style sheets*, you separate content (HTML) from formatting (CSS). That makes it easy to use the same styles for two or more pages.
- If you use *embedded styles*, you have to copy the styles to other documents before you can use them in those documents.
- If more than one declaration for the same property is applied to the same element, the last declaration overrides the earlier declarations.
- When you specify a relative URL for an external CSS file, the URL is relative to the current file.

Figure 1-13 How to provide the CSS styles for an HTML page

How to code the basic CSS selectors

Figure 1-14 shows how to code the basic CSS selectors for applying styles to HTML elements. To start, this figure shows the body of an HTML document that contains a main and a footer element. Here, the two <p> elements in the main element have class attributes with the value "blue". Also, the <p> element in the footer has an id attribute with the value "copyright" and a class attribute with two values: "blue" and "right". This means that this element is assigned to two classes.

The four style rules in the first group of CSS examples are type selectors. To code a type selector, you just code the name of the element. As a result, the first style rule in this group selects the body element. The second style rule selects the main element. The third style rule selects the h1 element. And the fourth style rule selects all <p> elements.

In these examples, the first style rule changes the font for the body, and all of the elements within the body inherit this change. This style rule also sets the width of the body and centers it in the browser. Then, the second style rule puts a border around the main element and puts some padding inside the border. It also makes the main element a block element. This is necessary for IE because IE doesn't treat the main element as a block element.

The third style rule that uses a type selector sets the margins for the heading. In this case, all the margins are set to zero except for the bottom margin. Last, the style rule for the paragraphs sets the margins for the top, bottom, and left side of the paragraphs. That's why the paragraphs in the main element are indented.

The style rule in the second group of CSS examples uses an id selector to select an element by its id. To do that, the selector is a pound sign (#) followed by the id value that uniquely identifies an element. As a result, this style rule selects the <p> element that has an id of "copyright". Then, its one property declaration sets the font-size for the paragraph to 90% of the default font size.

The two style rules in the last group of CSS examples use class selectors to select HTML elements by class. To do that, the selector is a period (.) followed by the class name. As a result, the first style rule selects all elements that have been assigned to the "blue" class, which are all three <p> elements. The second style rule selects any elements that have been assigned to the "right" class. That is the paragraph in the footer. Then, the first style rule sets the color of the font to blue and the second style rule aligns the paragraph on the right.

One of the key points here is that a class attribute can have the same value for more than one element on a page. Then, if you code a selector for that class, it will be used to format all the elements in that class. By contrast, since the id for an element must be unique, an id selector can only be used to format a single element.

As you probably know, there are several other selectors that you can use with CSS. But the ones in this figure will get you started. Then, whenever an application in this book requires other selectors, the selectors will be explained in detail.

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

```
<body>
    <main>
        <h1>The Speaker Lineup</h1>
        <p class="blue">October 19: Jeffrey Toobin</p>
        <p class="blue">November 16: Andrew Ross Sorkin</p>
    </main>
    <footer>
        <p id="copyright" class="blue right">Copyright SJV Town Hall</p>
    </footer>
</body>
```

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

Elements by type

```
body {
    font-family: Arial, Helvetica, sans-serif;
    width: 400px;
    margin: 1em auto; }
main {
    display: block;
    padding: 1em;
    border: 2px solid black; }
h1 { margin: 0 0 .25em; }
p { margin: .25em 0 .25em 3em; }
```

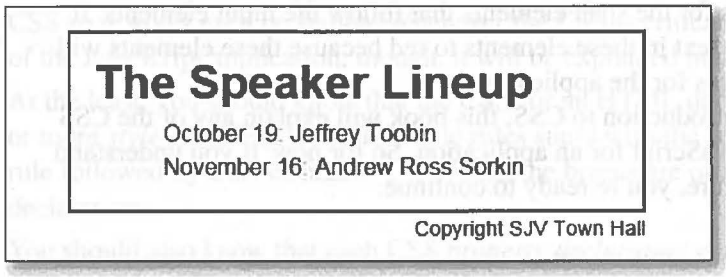
One element by ID

```
#copyright { font-size: 90%; }
```

Elements by class

```
.blue { color: blue; }
.right { text-align: right; }
```

The elements displayed in a browser



Description

- You code a selector for all elements of a specific type by naming the element. This is referred to as a *type selector*.
- You code a selector for an element with an id attribute by coding a pound sign (#) followed by the id value. This is known as an *id selector*.
- You code a selector for an element with a class attribute by coding a period followed by the class name. Then, the style rule applies to all elements with that class name. This is known as a *class selector*.

Figure 1-14 How to code the basic CSS selectors

How to code CSS style rules

Figure 1-15 presents the CSS for the Email List application that was presented earlier in this chapter. This is typical of the CSS for the applications in this book. Since the focus of this book is on JavaScript, not CSS, the CSS for the book applications is usually limited. For instance, the CSS in this example doesn't require id or class selectors.

Just to make sure we're using the same terminology, this CSS contains six *style rules*. Each style rule consists of a selector, a set of braces { }, and one or more *property declarations* within the braces. Also, each property declaration consists of a *property name*, a colon, the value or values for the rule, and an ending semicolon.

For instance, the first style rule is for the body element. It consists of six property declarations that set the font, background color, margins, width, border, and padding for the body. Here, the declaration for the margin property sets the top and bottom margins to zero and the right and left margins to "auto", which means the body will be centered in the browser window. And the declaration for the padding property sets the padding around the contents to 2 ems on the right and left and 1 em on the bottom. (An *em* is a typesetting term that is approximately equal to the width of a capital letter M.)

The second style rule is for the h1 element, and its one property declaration sets the color of the font to blue. Similarly, the one property declaration of the third style rule sets the bottom margin of div elements to 1 em.

The fourth style rule is for the label elements, and the fifth style rule is for the input elements. The style rule for label elements sets the display property to inline-block and the width property to 11 ems, and it aligns the text in the labels on the right. The style rule for input elements sets the left margin to 1 em, and it sets the right margin to .5 em.

The last style rule is for the span elements that follow the input elements. It just sets the color of the text in these elements to red because these elements will display the error messages for the application.

Beyond this brief introduction to CSS, this book will explain any of the CSS that is relevant to the JavaScript for an application. So for now, if you understand the style rules in this figure, you're ready to continue.

The CSS file for a typical application in this book

```
body {
    font-family: Arial, Helvetica, sans-serif;
    background-color: white;
    margin: 0 auto;
    width: 670px;
    border: 3px solid blue;
    padding: 0 2em 1em;
}
h1 {
    color: blue;
}
div {
    margin-bottom: 1em;
}
label {
    display: inline-block;
    width: 11em;
    text-align: right;
}
input {
    margin-left: 1em;
    margin-right: 0.5em;
}
span {
    color: red;
}
```

Description

- Because the focus of this book is JavaScript, not CSS, the CSS that's used in this book is usually simple. We just apply enough CSS to make each application look okay and work correctly.
- In fact, for most of the applications in this book, you won't have to understand the CSS so it won't even be shown. Whenever the CSS is critical to the understanding of the JavaScript application, though, it will be explained in detail.
- At the least, you should know that the CSS for an HTML document consists of one or more *style rules*. Each of these style rules starts with the selector for the style rule followed by a set of braces { }. Within the braces are one or more property declarations.
- You should also know that each CSS *property declaration* consists of a *property name*, a colon, the value or values for the property, and a semicolon.
- Many programmers use the popular Bootstrap CSS framework for their websites. Bootstrap's basic module allows you to lay out web pages using predefined classes. While beyond the scope of this book, Bootstrap is easy to use. You can search the Internet for more information.

Figure 1-15 How to code CSS style rules

How to test a JavaScript application

Next, you'll learn how to test a JavaScript application. To do that, you run the HTML for the web page that uses the JavaScript.

How to run a JavaScript application

When you develop a JavaScript application, you're usually working on your own computer or your company's server. Then, to run the application, you use one of the four methods shown in figure 1-16. Of the four, it's easiest to run the HTML page from the IDE or text editor that you're using to develop the HTML, CSS, and JavaScript files. You'll learn more about how to use VS Code to do that in a moment.

Otherwise, you can open the HTML file from your browser. To do that, you can press Ctrl+O to start the Open command. Or, you can find the file using File Explorer (Windows) or Finder (macOS) and then double-click on it. That will open the page in your system's default browser. Of course, you can also run a new page by clicking on the link to it in the current page.

After an application has been uploaded to an Internet web server, you can use the second set of methods in this figure to run the application. The first way is to enter a *Uniform Resource Locator* (*URL*) into the address bar of your browser. The second way is to click on a link in one web page that requests another page.

As the diagram in this figure shows, the URL for an Internet page consists of four components. In most cases, the *protocol* is HTTP or HTTPS. If you omit the protocol, the browser uses HTTP or HTTPS 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 often omit the "www." from the domain name.

The third component is the *path* where the file resides on the server. The path lists the folders 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 "/ourwork/".

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

The web page at javascript_jquery/book_apps/ch01/email_list/index.html

Four ways to run an HTML page that's on your own server or computer

- From your browser, use the Ctrl+O shortcut key combination to start the Open command. Then, browse to the HTML file and double-click on it.
- Use File Explorer (Windows) or Finder (macOS) to find the HTML file, and double-click on it.
- Use the features of your text editor or IDE.
- Click on a link in the current web page to load the next web page.

Two ways to run an HTML page that's on the Internet

- Enter the URL of the web page into the browser's address bar.
- Click on a link in the current web page to load the next web page.

The components of an HTTP URL on the Internet

```
http://www.modulemedia.com/ourwork/index.html
```

protocol | domain name | path | filename

What happens if you omit parts of a URL

- If you omit the protocol, the default of http:// or https:// 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.

Description

- When you are developing JavaScript applications, you usually store them on your own computer instead of the Internet. So when you test the applications, you run them from your own computer.
- Later, after the applications are deployed to your Internet web server, you can run the applications from the Internet.

Figure 1-16 How to run a JavaScript application

How to find errors in your code

As you test even the simplest of applications, you're likely to have errors in your code. When that happens, the JavaScript may not run at all, or it may run for a short while and then stop. That's why figure 1-17 shows you how to find the errors in your code.

As this figure shows, if a JavaScript application doesn't run or stops running, you start by opening the *developer tools*. Although there are several ways to do that, you'll use the F12 key most of the time. That's why the developer tools for Chrome and other browsers are often referred to as the *F12 tools*.

Next, you open the Console panel of the developer tools to see if there's an error message. In this figure, the console shows a message for an error that occurred when the user started the Email List application, clicked on the Join our List button, and nothing happened.

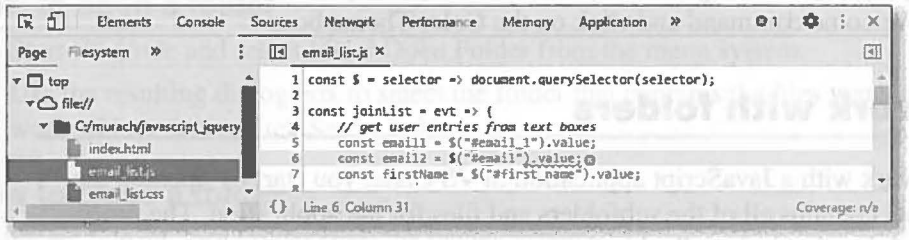
Then, if you click on the link to the right of the error message, the Sources panel is displayed with the JavaScript statement that caused the error highlighted. In this case, the problem is that "email" should be "email_2" since that's the id of the second text box in the HTML.

Note that the Sources panel in this figure doesn't show the Debugging pane that's typically displayed at the right side of the panel. If you're not using this pane, you can close it by clicking on the Hide Debugger button at the right above the JavaScript code. To redisplay it, you can click the Show Debugger button.

In chapter 5, you'll learn more about testing and debugging, but the technique shown in this figure will be all that you need until your applications get more complicated.

Chrome with an open Console panel that shows an error

The Sources panel after the link in the Console panel has been clicked



How to open or close Chrome's developer tools

- To open, press F12 or Ctrl+Shift+I. Or, click on the Menu button in the upper right corner of the browser, and select More Tools→Developer Tools.
- To close, click on the X in the upper right corner of the tools panel or press F12.

How to find the JavaScript statement that caused the error

- Open the Console panel by clicking on the Console tab. You should see an error message like the one above along with the line of code that caused the error.
- Click on the link to the right of the error message that indicates the line of code. That will open the Sources panel with the portion of JavaScript code that contains the statement displayed and the statement highlighted.

Description

- Chrome's *developer tools* provide some excellent debugging features, like identifying the JavaScript statement that caused an error.
- Because you usually start the developer tools by pressing the F12 key, these tools are often referred to as the *F12 tools*.

Figure 1-17 How to find errors in your code

How to use Visual Studio Code to develop JavaScript applications

Because HTML, CSS, and JavaScript are just text, you can use any text editor to create the files for a JavaScript application. However, a better editor or an *Integrated Development Environment* (*IDE*) can speed development time and reduce coding errors. For this book, we recommend Visual Studio Code (or VS Code). It is a free editor that runs on Windows, macOS, and Linux, and it can greatly improve your productivity.

To learn how to install VS Code, you can refer to appendix A (Windows) or appendix B (macOS). You can also learn how to use VS Code for the common development functions in the topics that follow. If you prefer to use another editor, you can skip these topics. But even then, you may want to browse these topics because they will give you a good idea of what an editor should be able to do. They may also encourage you to give VS Code a try.

Incidentally, the VS Code screenshots in the figures that follow use the light Visual Studio color theme. In contrast, the default theme for VS Code is dark. If you want to change the default theme on your system, you can use the Help→Welcome command and click on the Color Theme box.

How to work with folders

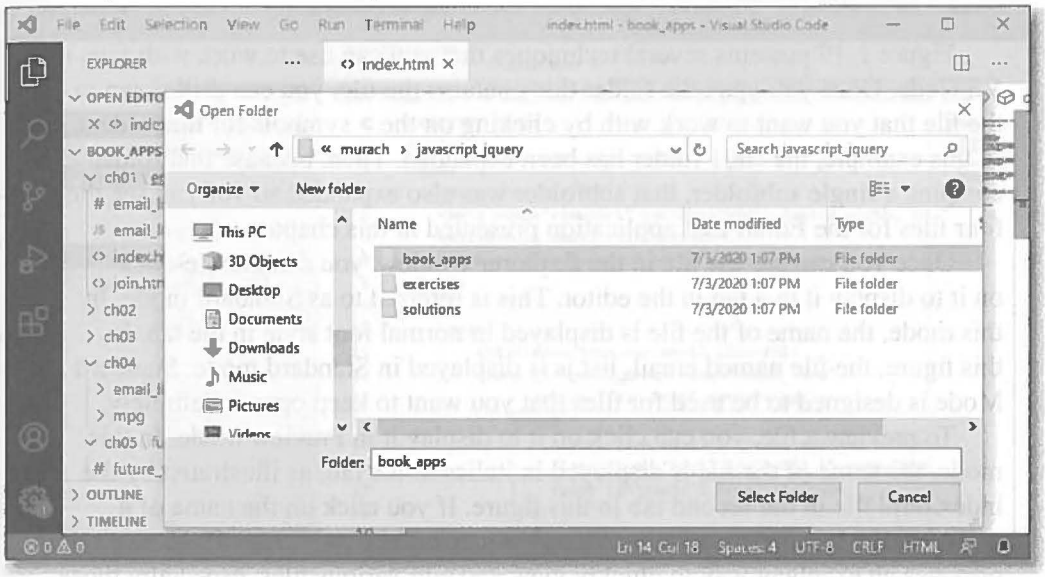
To work with a JavaScript application in VS Code, you start by opening the folder that contains all of the subfolders and files for the application. The procedure for doing that is described in figure 1-18.

After you open the folder for an application, it's subfolders and files are displayed in the Explorer window at the left side of VS Code's main window. In this example, the folder is for the Email List application presented in this chapter. If you open another folder, it replaces the folder that's currently open.

To make it easier to work with the applications for this book, we recommend that you use VS Code to open the folder that contains all of the applications. This is illustrated by the dialog box in this figure. After you do that, all of the chapter folders will be listed in the Explorer window, as you'll see in the next figure. Then, you'll be able to expand these folders to view and work with the subfolders and files that they contain. You'll get a chance to do this in exercise 1-3 at the end of this chapter.

This figure also shows how to add, rename, and delete folders. Note here that the technique for adding a folder to the main folder, in this case, the Email_List folder, is different from the technique for adding a folder to another folder. Finally, this figure shows how to close a folder.

The dialog box for choosing a folder in VS Code



How to open a folder

1. Start VS Code and select File→Open Folder from the menu system.
2. Use the resulting dialog box to select the folder that contains the files you want to work with and then click Select Folder.

How to close a folder

- Select File→Close Folder from the menu system.

How to add, rename, or delete a folder

- To add a folder to the main folder, point to the name of the folder in the Explorer window and click the New Folder icon that's displayed to its right. Then, enter a name for the folder.
- To add other folders, right-click on a folder in the Explorer window and select New Folder. Then, enter a name for the 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.

Description

- *Visual Studio Code* (also known as *VS Code*) is a text editor that you can use to work with a variety of file types, including JavaScript, HTML, and CSS.
- All of the files for the applications presented in this book are stored in this folder: **`\murach\javascript_jquery\book_apps`**

Figure 1-18 How to work with folders in VS Code

How to work with files

Figure 1-19 presents several techniques that you can use to work with files in VS Code. Once you open the folder that contains the file, you can drill down to the file that you want to work with by clicking on the > symbols for the folders. In this example, the ch01 folder has been expanded. Then, because that folder contains a single subfolder, that subfolder was also expanded so you can see the four files for the Email List application presented in this chapter.

Once you can see the file in the Explorer window, you can double-click on it to display it in a tab in the editor. This is referred to as Standard mode. In this mode, the name of the file is displayed in normal font style in the tab. In this figure, the file named email_list.js is displayed in Standard mode. Standard Mode is designed to be used for files that you want to keep open indefinitely.

To preview a file, you can click on it to display it in Preview mode. In this mode, the name of the file is displayed in italics in the tab, as illustrated by the index.html file in the second tab in this figure. If you click on the name of a different file, VS Code loads that file in the same tab of the editor. Preview mode provides an excellent way to quickly view and edit various files, especially those files that you don't want to keep open for a long time.

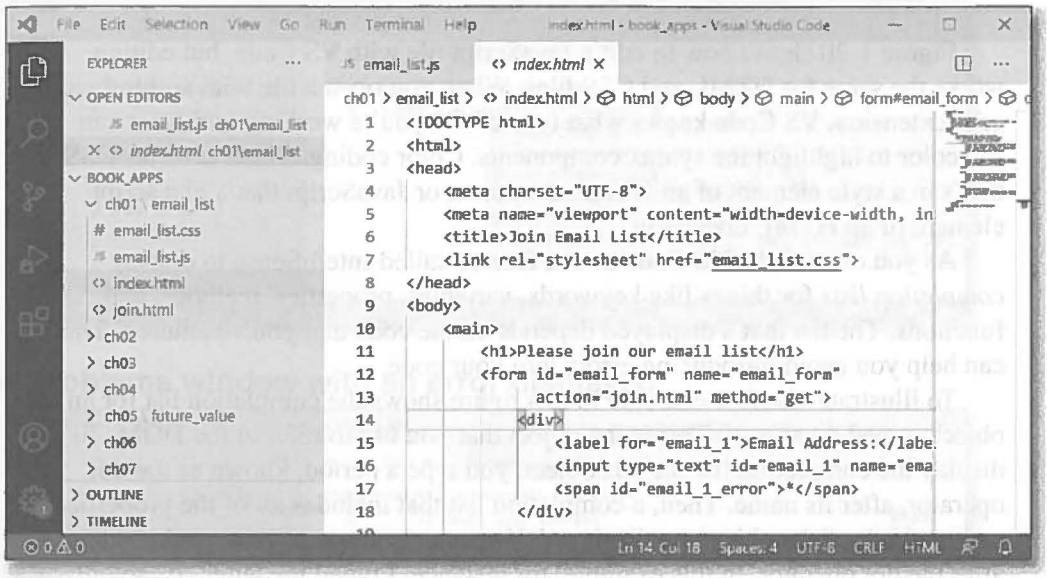
When two or more files are displayed in an editor, you can switch between them by clicking on the tab for the file you want to display. Alternatively, you can select a file from the Open Editors list at the top of the Explorer window. This list includes all of the files that are currently displayed in an editor.

To close a file, you can use one of the techniques described in this figure. If you try to close a file and it has changes that haven't been saved, you'll be asked if you want to save the changes. You can also save changes to one or more files without closing them as described in this figure. You might want to do that if you're making more than just a few changes and you want to be sure the changes you've already made aren't lost.

This figure ends by showing how to add, rename, or delete a file. The skills for doing that are identical to the skills for adding, renaming, or deleting a folder, except that when you add a file, you click the New File icon or select New File. Note that when you add a file, you must include an extension (.html, .css, or .js) depending on whether it is going to be an HTML, CSS, or JavaScript file. Then, VS Code will know what type of file it is, and its editor will be adjusted to the syntax of that type of file when it is opened.

When you add a new file, you should realize that VS Code doesn't generate any code for it. Because of that, you have to enter all of the code yourself. Another option, though, is to copy code from another file and paste it into the new file. Or, you can open a similar file and then select File→Save As to save it with a new name. Then, you can delete the code you don't need and add code you do need.

VS Code with files in Standard and Preview mode



How to preview or open a file

- To open a file, double-click on it in the Explorer window. This displays the file in a tab in the editor with the name of the file in normal font style, indicating that you are in Standard Mode.
- To preview a file, click on it in the Explorer window. This displays the file in a tab in the editor with the name of the file in italics, indicating that you are in Preview Mode. If you open or preview another file, VS Code reuses the tab.
- To display a file that's already open, click on its tab or select it from the Open Editors list at the top of the Explorer window.

How to close or save a file

- To close a file, click the X in the upper right corner of the tab for the file, click the X to the left of the file name in the Open Editors list, or select File→Close Editor.
- If you close a file with changes, you'll be asked if you want to save the changes.
- If you want to saves changes without closing a file, select File→Save. To save changes to more than one file, select File→Save All.

How to add, rename, or delete a file

- To add a file, you use the same skills as you do for adding a folder except that you click the New File icon or select New File. When you name the file, be sure to include an extension.
- VS Code doesn't generate any starting code for new files. As a result, you must enter all code for the file yourself or use similar code from another file.
- To rename or delete a file, you use the same skills as you do for renaming or deleting a folder.

Figure 1-19 How to work with files in VS Code

How to edit a JavaScript file

Figure 1-20 shows how to edit a JavaScript file with VS Code, but editing works the same for HTML and CSS files. When you open a file with an html, css, or js extension, VS Code knows what type of file you're working with so it can use color to highlight the syntax components. Color coding is also used for CSS that's in a style element of an HTML document or JavaScript that's in a script element of an HTML document.

As you enter code, VS Code uses a feature called IntelliSense to display *completion lists* for things like keywords, variables, properties, methods, and functions. The list that's displayed depends on the code that you've entered. That can help you avoid introducing errors into your code.

To illustrate, the first example in this figure shows the completion list for an object named document. This is the object that you use to refer to the DOM. To display the completion list for this object, you type a period, known as the dot operator, after its name. Then, a completion list that includes all of the properties and methods of the object are displayed. You can also type additional characters after the dot operator. In this example, for instance, I typed the letter "q" so only the properties and methods that contain this letter are displayed.

When a completion list is displayed, you can insert an item by clicking on it. Alternatively, you can highlight the entry you want by using the arrow keys to scroll through the list. Then, you can complete the entry by pressing the Tab or Enter key.

This also works with HTML and CSS entries. If, for example, you type "<s" in an HTML document, VS Code presents a list of the elements that start with the letter *s*. Then, if you select one of the elements and enter a right angle bracket, VS Code adds the closing tag. This feature also works when you start an attribute.

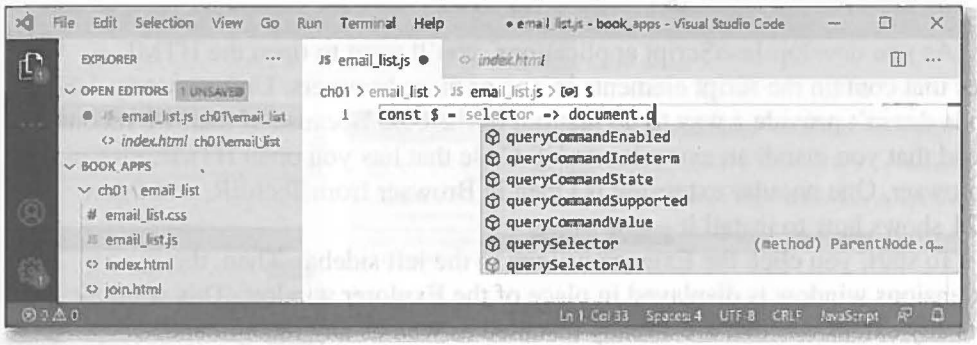
Similarly, if you start to enter the name of an element for a CSS style rule, VS Code presents a list of the elements with the letters you enter. If you enter one or more letters to start a property declaration, VS Code presents a list of the properties that start with those letters. And if you start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

If VS Code detects an error as you type, it displays a red wavy line under the error as shown in the second example. Here, the else keyword is underlined because the preceding if block doesn't end with a right brace (}). To display the description of an error, you can hover the mouse over the red 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.

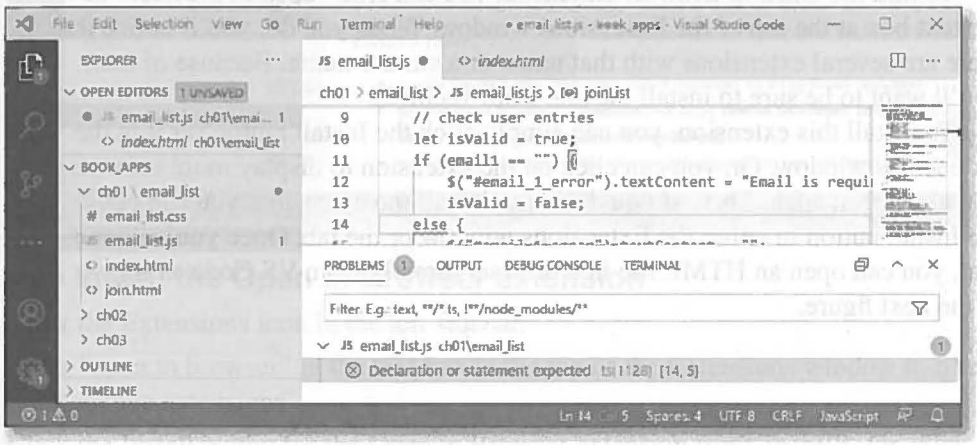
The Explorer window also indicates if a file contains errors. To do that, it displays the name of the file, as well as the name of the folder that contains the file, in red. In addition, it displays the number of errors in the file to the right of the filename.

As you work with VS Code, you'll see that it has the same type of interface that you've used with other programs. So if you want to do something that isn't presented in this chapter, try right-clicking on an item to see what menu options are available and see what's available from the menu system. With a little experimentation, you'll find that this program is easy to use.

The completion list for selecting a property or method of an object



The Problems window with an error displayed



How to use the IntelliSense feature

- IntelliSense displays *completion lists* for things like keywords, variables, properties, methods, and functions as you type so you can enter them correctly.
- To insert an item from a completion list, click on it or highlight it and then press the Tab or Enter key.
- If you enter an opening parenthesis or brace, the closing parenthesis or brace is added automatically.

How to identify the errors that are marked by VS Code

- If VS Code detects a syntax error, it underlines it with a red wavy line.
- To get the description for an error, hover the mouse over the red 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 simplifies the task of editing JavaScript code by providing features like IntelliSense, error checking, and color coding.

Figure 1-20 How to edit a JavaScript file in VS Code

How to install the Open in Browser extension

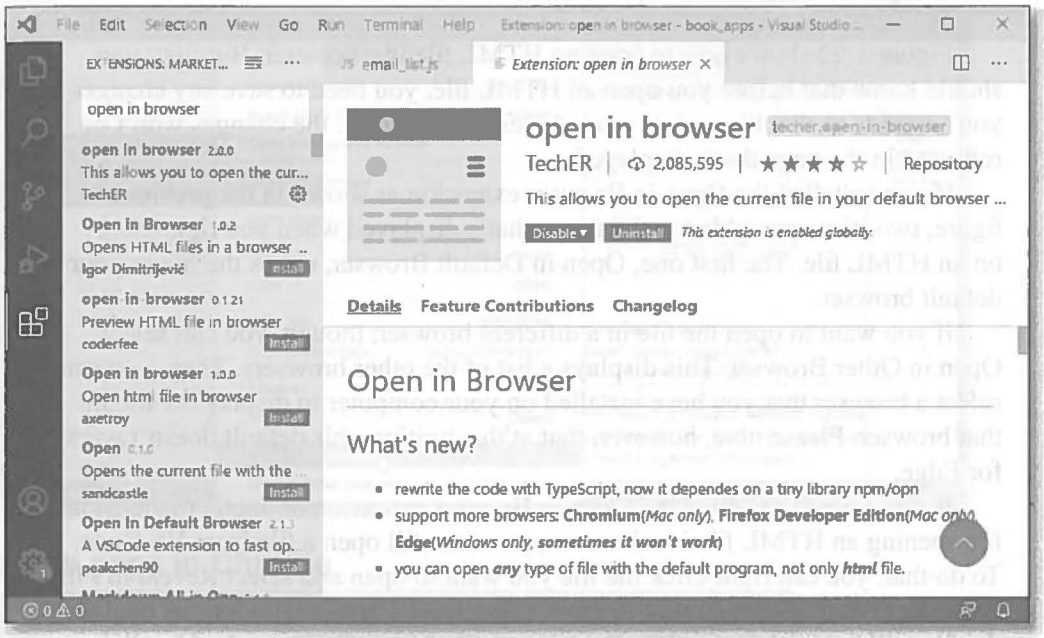
As you develop JavaScript applications, you'll want to open the HTML files that contain the script elements in one or more browsers. Unfortunately, VS Code doesn't provide a way to do that out of the box. Because of that, we recommend that you install an extension to VS Code that lets you open HTML files in a browser. One popular extension is Open in Browser from TechER, and figure 1-21 shows how to install it.

To start, you click the Extensions icon in the left sidebar. 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 Open in Browser extension, you can type "open in browser" in the text box at the top of the Extensions window. When you do, you'll notice that there are several extensions with that name or a similar name. Because of that, you'll want to be sure to install the one from TechER.

To install this extension, you can simply click the Install button for it in the Extensions window. Or, you can click on the extension to display more information about it in a tab. Then, if you decide to install the extension, you can click the Install button in either the Extensions window or the tab. Once you've done that, you can open an HTML file in a browser directly from VS Code as shown in the next figure.

VS Code as the Open in Browser extension is installed



How to install the Open in Browser extension

1. Click the Extensions icon in the left sidebar.
2. Enter "open in browser" in the text box at the top of the Extensions window to filter the available extensions.
3. Click the Install button for the Open in Browser extension from TechER. Or, click on the extension to display information about it in a tab, and then click the Install button in the tab.

Description

- By default, VS Code doesn't provide a way to open an HTML file in a browser so you can test the JavaScript code it uses.
- To make it easier to open an HTML file in a browser, you can install an extension to VS Code. One popular extension is the Open in Browser extension developed by TechER.

Figure 1-21 How to install the Open in Browser extension

How to open an HTML file in a browser

Figure 1-22 shows how to open an HTML file in a browser. But first, you should know that before you open an HTML file, you need to save any changes you've made to that file and its related files. If you don't, the changes won't be reflected in the page that's displayed.

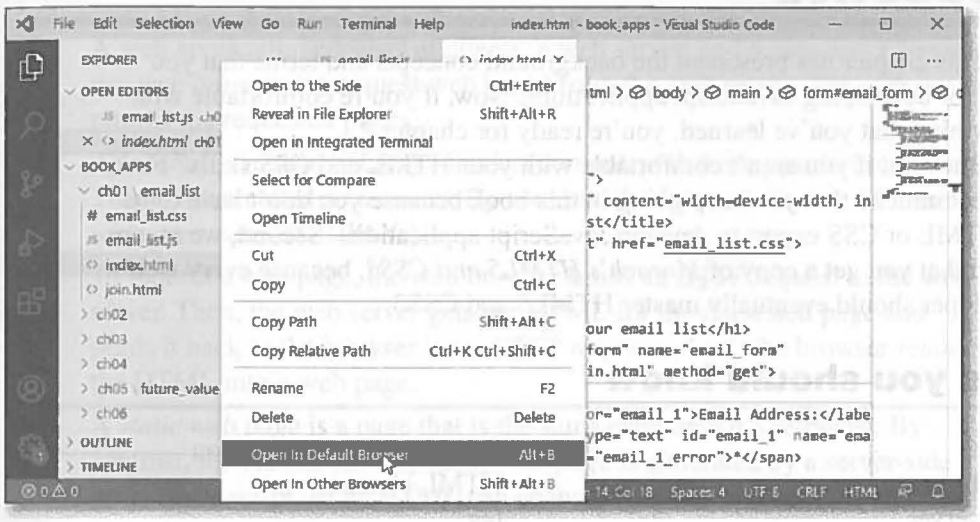
If you installed the Open in Browser extension as shown in the previous figure, two items are added to the menu that's displayed when you right-click on an HTML file. The first one, Open in Default Browser, opens the file in your default browser.

If you want to open the file in a different browser, though, you can select Open in Other Browser. This displays a list of the other browsers. Then, you can select a browser that you have installed on your computer to display the file in that browser. Please note, however, that at this writing, this default doesn't work for Edge.

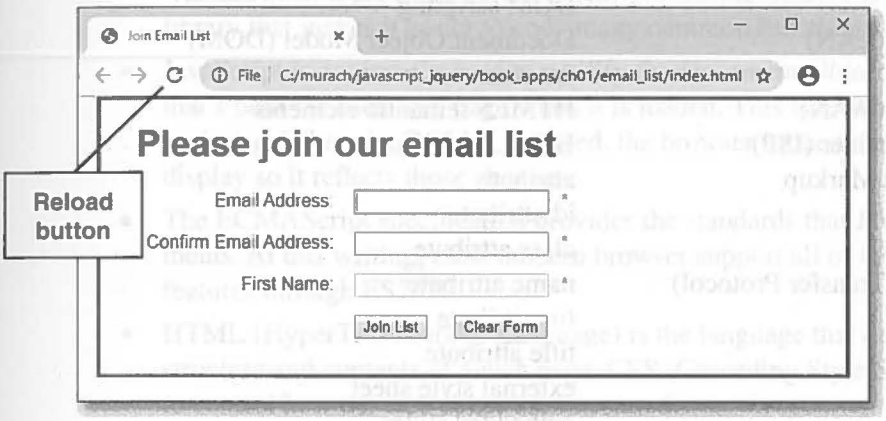
If you haven't installed the Open in Browser extension or another extension for opening an HTML file in a browser, you can still open a file from VS Code. To do that, you can right-click the file you want to open and select Reveal in File Explorer (Windows) or Reveal in Finder (macOS). Then, the folder that contains the file will be displayed in File Explorer or Finder, and you can double-click on the file to open it.

When you open an HTML file from File Explorer or Finder or by using the Open in Browser extension, a new browser or browser tab is opened each time. But in most case, that's not what you want. To avoid that, you can open the file just once. Then, after you find and fix the errors in VS Code, you can save the changes, switch to the browser, and click on the Reload or Refresh button in the browser to reload the file with the changes. That way, you use the same tab or Browser instance each time you test the application.

How to open an HTML file using the Open in Browser extension



The web page in Chrome



Description

- If you installed the Open in Browser extension as shown in the previous figure, you can open an HTML file by right-clicking on the file in the Explorer window and selecting Open in Default Browser to open the file in your default browser.
- If you select Open in Other Browsers when you're using the Open in Browser extension, you can select the browser in which you want the file to be opened. Note, however, that at this writing, this doesn't work for Edge.
- If you haven't installed the Open in Browser extension, you can right-click on the file in the Explorer window and select Reveal in File Explorer (Windows) to display it in File Explorer or Reveal in Finder (macOS) to display it in Finder. Then, you can double-click the file to open it.
- Every time you open an HTML file from VS Code, another browser instance or browser tab is opened. Another alternative is to save the corrected files in VS Code, switch to the browser, and click its Reload or Refresh button.

Figure 1-22 How to open an HTML file in a browser

Perspective

This chapter has presented the background concepts and terms that you need for developing JavaScript applications. Now, if you're comfortable with everything that you've learned, you're ready for chapter 2.

But what if you aren't comfortable with your HTML and CSS skills? First, we recommend that you keep going in this book because you don't have to be an HTML or CSS expert to develop JavaScript applications. Second, we recommend that you get a copy of *Murach's HTML5 and CSS3*, because every web developer should eventually master HTML5 and CSS3.

Terms you should know

client
web browser
web server
network
intranet
local area network (LAN)
Internet
wide area network (WAN)
Internet service provider (ISP)
HTML (HyperText Markup Language)
static web page
HTTP (HyperText Transfer Protocol)
HTTP request
HTTP response
render a web page
dynamic web page
application server
database server
server-side processing
round trip
JavaScript
JavaScript engine
scripting language
client-side processing
image swap
image rollover
data validation
jQuery
European Computer Manufacturers Association (ECMA)
ECMAScript specification
HTML element
tag
CSS (Cascading Style Sheets)
DOM scripting
Document Object Model (DOM)
function
HTML5 semantic elements
HTML5 semantics
attribute
id attribute
class attribute
name attribute
for attribute
title attribute
external style sheet
embedded styles
CSS selector
type selector
id selector
class selector
style rule
property declaration
property name
URL (Uniform Resource Locator)
protocol
domain name
path
developer tools
IDE (Integrated Development Environment)

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, the *Internet* consists of many *wide area networks* (*WANs*).
- To request a web page, the web browser sends an *HTTP request* to the web server. Then, the web server gets 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* is a page that is the same each time it's retrieved. By contrast, the HTML for a *dynamic web page* is generated by a server-side program or 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*. *jQuery* is a JavaScript library that makes it easier to code many common functions.
- JavaScript is commonly used to modify the *Document Object Model* (*DOM*) that's built for each web page when it is loaded. This is referred to as *DOM scripting*. When the DOM is changed, the browser immediately changes its display so it reflects those changes.
- The ECMAScript specification provides the standards that JavaScript implements. At this writing, most modern browser support all of the important features through ES2020.
- HTML (HyperText Markup Language) is the language that defines the structure and contents of a web page. CSS (Cascading Style Sheets) is used to control how the web pages are formatted.
- You can view a web page that's on your own computer or server or on an Internet server. To view a web page on an Internet server, you can enter the URL (Uniform Resource Locator) that consists of the protocol, domain name, path, and filename into a browser's address bar.
- To help find errors when you test a JavaScript application, you can use the Console panel in Chrome's developer tools.
- To develop JavaScript applications, you can use an Integrated Development Environment (IDE) or a text editor like VS Code.

Before you do the exercises for this book...

Before you do the exercises for this book, you should download and install the Chrome browser as well as the applications for this book. If you're going to use VS Code, you should also download and install that product. The procedures for installing the software and applications for this book are in appendix A (Windows) and appendix B (macOS).

Exercise 1-1 Run the Email List application

In this exercise, you'll run the Email List application that's presented in figures 1-7 through 1-9.

Open the application in Chrome

1. Start Chrome if it isn't already open. Then, use the Ctrl+O key combination to open this HTML file:
   murach\javascript_jquery\book_apps\ch01\email_list\index.html
2. To test what happens when you don't enter any data, just click the Join List button without entering any data.
3. Enter an email address in the first text box and invalid data in the second text box and click the Join List button to see what error messages are displayed.
4. Enter valid data for all three text boxes and click on the button. Then, the data is submitted for processing and a new web page is displayed.

Preview the developer tools

5. To rerun the application, click the browser's Back button. This should reset the entries.
6. Enter an email address in the first text box, leave the other text boxes empty, and click the Join List button. Then, press F12 to open Chrome's developer tools. Display the Console tab by clicking on it, and note that it doesn't contain any error messages.
7. Click on the Sources tab. Then, if it's not already displayed, click on the email_list.js file in the left pane to see the JavaScript code for the page. Do the same for the index.html file to see the HTML for the page.
8. Click on the Elements tab, and drill down to see the HTML elements within the form element for the page. In the span elements, you can see the error messages that are displayed, not the starting values in the HTML. That's because the Elements tab represents the Document Object Model for the page, and JavaScript has changed that model.
9. Do more experimenting if you want. These are powerful debugging tools that you'll use a lot as you go through this book, so it's good to become familiar with them. When you're through, press the F12 button to close the tools.

Exercise 1-2 Run other section 1 applications

This exercise has you run two of the applications presented in chapters 6 and 7. That will give you some idea of what you'll be able to do when you complete this section.

Run the FAQs application of chapter 6

1. Open this file in the Chrome browser:
   `murach\javascript_jquery\book_apps\ch06\faqs\index.html`
2. Click on the first heading to display the text for it, and click the heading again to hide the text.
3. Tab to the next heading and press the Enter key to display the text for it, and press the Enter key again to hide the text.

Run the Image Swap application of chapter 7

4. Open this file in the Chrome browser:
   `murach\javascript_jquery\book_apps\ch07\image_swap\index.html`

   Notice the four small images near the top of the page, and notice that a larger version of the first image is displayed below the smaller images.
5. Click on the second image to see that a larger version of that image is now displayed.
6. Tab to the next image and press the Enter key to display a larger version of that image.

Exercise 1-3 Get started with VS Code

This exercise is for readers who are going to use VS Code with this book. It guides you through the process of using VS Code to work with the files for the JavaScript book applications and exercises that you've downloaded. Before you do this exercise, you need to install VS Code as shown in Appendix A or B.

Start VS Code and consider changing the color theme

1. Start VS Code. Then, if the Welcome page isn't displayed, use the Help→Welcome command to open it. Now, click on the Color Theme box and review the color themes. If you want to change the theme, do that now.

Open the folders for this book

2. Use the procedure in figure 1-18 to open the book applications that are stored in this folder:
   `murach\javascript_jquery\book_apps`

   After you open this folder, you should see the chapter folders within this folder listed in the Explorer window.

3. Use the same procedure to open the exercise starts that are stored in this folder:

   ```
   javascript_jquery\exercises
   ```

   When you do that, notice that the folder that contains the book applications is closed.

Test the Email List application

4. Use the procedure in figure 1-21 to install the Open in Browser extension.
5. In the Explorer window, click on the > symbol before ch01 to display the files for the Email List application.
6. Right-click the file named index.html, and select Open in Default Browser to open that file in your default browser.
7. Switch back to VS Code. If you have another browser installed on your computer, right-click on the index.html file again, but this time select Open in Other Browsers. Then, click on the other browser to open the file in that browser. But remember that at this writing, this doesn't work for Edge.

Edit the JavaScript code

8. In the Explorer window, double-click on the file named email_list.js to open that file, and note the colors that are used for syntax highlighting.
9. In the JavaScript file, delete the right parenthesis in the first line of code. This should display a red wavy line under the semicolon at the end of the line. Then, hover the mouse over this line to display the error description. This illustrates VS Code's error-checking feature. Now, undo the change that you made. (To undo a change with the keyboard, press Ctrl+Z.)
10. In the JavaScript file, after the third statement that starts with const in the joinList() function, start a statement on a new line with these characters:

    ```
    if (e
    ```

    This should insert the closing parenthesis and display a list of the possible entries that start with the letter e. Here, you can see that email1 and email2 are included in the list. These are the constants that are created by the first two const statements in this function, and this illustrates VS Code's IntelliSense feature. Now, undo this change.
11. Enter the statement that follows on a new line that comes right before the last line of the JavaScript code, which consists of just a right brace followed by a right parenthesis and a semicolon:

    ```
    alert("The DOM has now been built");
    ```

12. To test the statement that you added, select File→Save from the menu system to save your changes. Then, switch to your browser and click on the Reload or Refresh button. When you do, a dialog box should be displayed indicating that the DOM has been built. To close this dialog box, click on its OK button.
13. If you're curious, do more experimenting. Then, exit from VS Code and close the browser or tab.