Jennifer Kyrnin
Julie C. Meloni

THIRD EDITION

Sams Teach Yourself

HTML, CSS, and JavaScript

All in One

Pearson

Jennifer Kyrnin
Julie Meloni

Sams **Teach Yourself**

HTML, CSS, and JavaScript

Third Edition

All in One

Pearson

Sams Teach Yourself HTML, CSS, and JavaScript All in One, Third Edition

ISBN-13: 978-0-672-33808-3

ISBN-10: 0-672-33808-4

Library of Congress Control Number: 2018953965

01 18

Trademarks

Warning and Disclaimer

Editor-in-Chief
Mark Taub

Editor
Mark Taber

Managing Editor
Sandra Schroeder

Senior Project Editor
Lori Lyons

Copy Editor
Kitty Wilson

Project Manager
Suganya Karuppasamy

Indexer
Ken Johnson

Proofreader
Abigail Manheim

Technical Editor
Julie Meloni

Editorial Assistant
Cindy Teeters

Cover Designer
Chuti Prasertsith

Compositor
codemantra

Contents at a Glance

Table of Contents

Part III: Advanced Web Page Design with CSS

About the Authors

Jennifer Kyrnin consults professionally on web design and web development. She has built and maintained websites of all sizes, from small single-page sites to large million-page database-driven sites for international audiences. She has taught HTML, XML, and web design online since 1997 and is the author of *Sams Teach Yourself HTML5 Mobile Application Development*, *Sams Teach Yourself Responsive Web Design*, and *Sams Teach Yourself Bootstrap.*

Julie Meloni is a software development manager and technical consultant living in Washington, DC. She has written several books and articles on web-based programming languages and database topics, including the bestselling *Sams Teach Yourself PHP, MySQL, and Apache All in One*.

Dedication

To Mark and Jaryth. As usual, you inspired me and kept me writing.

Acknowledgments

I would like to thank Mark Taber for thinking of me when this new edition came out. I couldn't have had a good-looking book without Kitty Wilson, my extraordinary copy editor. Any errors are not her fault. And thanks also go to Julie Meloni for believing that I could take over her book effectively. I hope I live up to her reputation.

LESSON 4

Understanding JavaScript

What You'll Learn in This Lesson:

- What web scripting is and what it's good for
- How scripting and programming are different (and similar)
- What JavaScript is and where it came from
- How to include JavaScript commands in a web page
- What JavaScript can do for your web pages
- Beginning and ending scripts
- Formatting JavaScript statements
- How a script can display a result
- Including a script within a web document
- Testing a script in a browser
- Modifying a script
- Dealing with errors in scripts
- Moving scripts into separate files

The World Wide Web (WWW) began as a text-only medium; the first browsers didn't even support images within web pages. The Web has come a long way since those early days. Today's websites include a wealth of visual and interactive features in addition to useful content: graphics, sounds, animation, and video. Using web scripting languages, such as JavaScript, is one of the easiest ways to spice up a web page and to interact with users in new ways.

The first part of this lesson introduces the concept of web scripting and the JavaScript language. As the lesson moves ahead, you'll learn how to include JavaScript commands directly in your HTML documents and how your scripts will be executed when the page is viewed in a browser. You will work with a simple script, edit it, and test it in your browser, all the while learning the basic tasks involved in creating and using JavaScript scripts.

Learning Web Scripting Basics

You already know how to use two types of computer languages: HTML and CSS. You use HTML tags to describe how you want your document formatted. Then you use CSS to describe how you want the document displayed, and the browser shows the decorated content to the user. But because HTML and CSS are simple text-based languages, they can't respond to the user, make decisions, or automate repetitive tasks. Interactive tasks such as these require a more sophisticated language: a programming language or a *scripting* language.

Although many programming languages are complex, scripting languages are generally simple. They have a simple syntax, can perform tasks with a minimum of commands, and are easy to learn. JavaScript is a web scripting language that enables you to combine scripting with HTML and CSS to create interactive web pages.

Scripts and Programs

A movie or a play follows a script—a list of actions (or lines) for the actors to perform. A web script provides the same type of instructions for the web browser. A script in JavaScript can range from a single line to a full-scale application. (In either case, JavaScript scripts usually run within a browser.)

Some programming languages must be *compiled*, or translated, into machine code before they can be executed. JavaScript, on the other hand, is an *interpreted* language: The browser executes each line of script as it comes to it.

There is one main advantage to interpreted languages: Writing or changing a script is very simple. Changing a JavaScript script is as easy as changing a typical HTML document, and the change is enacted as soon as you reload the document in the browser.

NOTE

Interpreted languages have disadvantages, too: They can't execute really quickly, so they're not ideally suited for complicated work, such as graphics, and they require the interpreter (in JavaScript's case, usually a browser) in order to work.

Introducing JavaScript

JavaScript was developed in 1995 by Netscape Communications Corporation, the maker of the long-defunct Netscape web browser. JavaScript was the first web scripting language to be supported by browsers, and it is still by far the most popular.

NOTE

A bit of history: JavaScript was originally called LiveScript, and it was first introduced in Netscape Navigator 2.0 in 1995. It was soon renamed JavaScript to indicate a marketing relationship with Sun's Java language, although there is no other relationship, structurally or otherwise, between Java and JavaScript.

JavaScript is almost as easy to learn as HTML, and it can be included directly in HTML documents. Here are a few of the things you can do with JavaScript:

- Display messages to the user as part of a web page, in the browser's status line, or in alert boxes
- Validate the contents of a form and make calculations (for example, by having an order form automatically display a running total as you enter item quantities)
- Animate images or create images that change when you move the mouse over them
- Create ad banners that interact with the user rather than simply displaying a graphic
- Detect what browser is in use or what features the browser has and perform advanced functions only on browsers that support them
- Detect installed plug-ins and notify the user if a plug-in is required
- Modify all or part of a web page without requiring the user to reload it
- Display or interact with data retrieved from a remote server

You can do all this and more with JavaScript, including creating entire applications. We'll explore the uses of JavaScript throughout these lessons.

How JavaScript Fits into a Web Page

By using the `<script>` tag, as shown in Listing 4.1, you can add a short script (in this case, just one line) to a web document. The `<script>` tag tells the browser to start treating the text as a script, and the closing `</script>` tag tells the browser to return to HTML mode. In most cases, you can't use JavaScript statements in an HTML document except within `<script>` tags. The exception is event handlers, described later in this lesson.

LISTING 4.1 A Simple HTML Document with a Simple Script

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>The American Eggplant Society</title>
  </head>
```

```
  <body>
    <h1>The American Eggplant Society</h1>
    <p>Welcome to our site. Unfortunately, it is still
    under construction.</p>
    <p>We last worked on it on this date:
    <script>
    <!-- // Hide the script from old browsers
    document.write(document.lastModified);
    // Stop hiding the script -->
    </script>
    </p>
  </body>
</html>
```

JavaScript's `document.write` statement, which you'll learn more about later, sends output as part of the web document. In this case, it displays the modification date of the document, as shown in Figure 4.1.

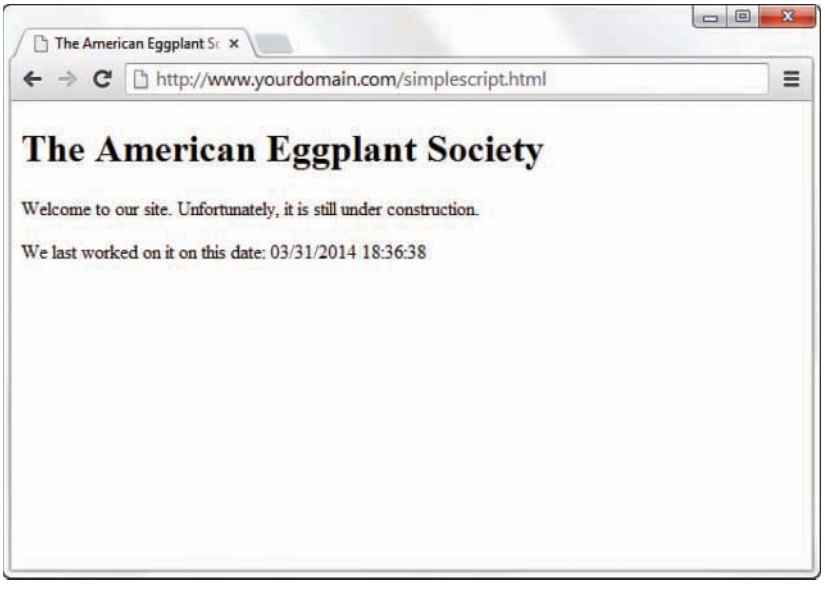

FIGURE 4.1
Using `document.write` to display a last-modified date.

In this example, we placed the script within the body of the HTML document. There are actually four places where you might use scripts:

- **In the body of the page**—In this case, the script's output is displayed as part of the HTML document when the browser loads the page.

- **In the header of the page, between the `<head>` tags**—Scripts in the header should not be used to create output within the `<head>` section of an HTML document, since that would likely result in poorly formed and invalid HTML documents, but these scripts can be referred to by other scripts here and elsewhere. The `<head>` section is often used for functions—groups of JavaScript statements that can be used as a single unit. You will learn more about functions in Lesson 20, "Getting Started with JavaScript Programming."
- **Within an HTML tag, such as `<body>` or `<form>`**—This is called an *event handler*, and it enables the script to work with HTML elements. When using JavaScript in event handlers, you don't need to use the `<script>` tag. You'll learn more about event handlers in Lesson 20.
- **In a separate file entirely**—JavaScript supports the use of files with the `.js` extension containing scripts; these can be included by specifying a file in the `<script>` tag. While using the `.js` extension is a convention, scripts can actually have any file extension, or none.

As you'll learn in Lesson 25, "JavaScript Best Practices," the best place to put JavaScript is inside the `<body>` tag, just before the closing `</body>` tag. This ensures that JavaScript is the last thing to load and so doesn't disrupt the speed of the rest of the page displaying.

Using Separate JavaScript Files

When you create more complicated scripts, you'll quickly find that your HTML documents become large and confusing. To avoid this problem, you can use one or more external JavaScript files. These are files with the `.js` extension that contain JavaScript statements.

External scripts are supported by all modern browsers. To use an external script, you specify its filename in the `<script>` tag, as shown here:

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

NOTE

The `type` attribute used to be required. But in HTML5 it can be left out if the script referenced is JavaScript.

Because in this case you'll be placing the JavaScript statements in a separate file, you don't need anything between the opening and closing `<script>` tags; in fact, anything between them will be ignored by the browser.

You can create the `.js` file by using the same text editor you use to write HTML and CSS. This file should contain one or more JavaScript commands and only JavaScript; it should not include `<script>` tags, other HTML tags, CSS, or HTML comments. Save the `.js` file in the same directory as the HTML documents that refer to it.

NOTE

External JavaScript files have a distinct advantage: You can link to the same `.js` file from two or more HTML documents. The browser stores this file in its cache, which can reduce the time it takes your web pages to display. But remember that every linked file requires an additional request to the server. So try to keep all your scripts in as few files as you can.

Understanding JavaScript Events

Many of the useful things you can do with JavaScript involve interacting with the user, and that means responding to *events*—for example, a link or a button being clicked. You can define event handlers within HTML tags to tell the browser how to respond to an event. For example, Listing 4.2 defines a button that displays a message when clicked.

LISTING 4.2 A Simple Event Handler

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Event Test</title>
  </head>
  <body>
    <h1>Event Test</h1>
    <button type="button"
            onclick="alert('You clicked the button.')">
            Click Me!</button>
  </body>
</html>
```

In various places throughout these lessons, you'll learn more about JavaScript's event model and how to create simple and complex event handlers.

Exploring JavaScript's Capabilities

If you've spent any time browsing the Web, you've undoubtedly seen lots of examples of JavaScript in action. The following sections provide some brief descriptions of typical applications for JavaScript, all of which you'll explore in later lessons.

Improving Navigation

Some of the most common uses of JavaScript are in navigation systems for websites. You can use JavaScript to create a navigation tool—for example, a drop-down menu to select the next page to read or a submenu that pops up when you hover over a navigation link.

When it's done right, this kind of JavaScript interactivity can make a site easier to use, even for browsers that don't support JavaScript.

Validating Forms

Form validation is another common use of JavaScript, although the form validation features of HTML5 have stolen a lot of JavaScript's thunder here as well. A simple script can read values the user types into a form and make sure they're in the right format, such as with zip codes, phone numbers, and email addresses. This type of client-side validation enables users to fix common errors without waiting for a response from the web server, telling them that their form submission was invalid. You'll learn how to work with form data in Lesson 27, "Working with Web-Based Forms."

Special Effects

One of the earliest and most annoying uses of JavaScript was to create attention-getting special effects—for example, scrolling a message in the browser's status line or flashing the background color of a page.

These techniques have fortunately fallen out of style, but thanks to the W3C DOM and the latest browsers, some more impressive effects are possible with JavaScript—for example, creating objects that can be dragged and dropped on a page or creating fading transitions between images in a slideshow. Some developers have HTML5, CSS3, and JavaScript working in tandem to create fully functioning interactive games.

Remote Scripting (AJAX)

For a long time, the biggest limitation of JavaScript was that there was no way for it to communicate with a web server. For example, you could use JavaScript to verify that a phone number had the right number of digits but not to look up the user's location in a database based on the number.

Now that most browsers support some of JavaScript's advanced features, this is no longer the case. Your scripts can get data from a server without loading a page, or they can send data back to be saved. These features are collectively known as AJAX (Asynchronous JavaScript and XML), or *remote scripting*.

Displaying Time with JavaScript

One common use of JavaScript is to display dates and times in the browser, and that's where we'll start putting some scripting pieces together. Because JavaScript runs on the browser, the times it displays will be in the user's current time zone. However, you can also use JavaScript to calculate "universal" (UTC) time.

NOTE

UTC, which stands for Universal Time (Coordinated), is the atomic time standard based on the old GMT (Greenwich Mean Time) standard. This is the time at the prime meridian, which runs through Greenwich, London, England.

Your script, like most other JavaScript programs, begins with the HTML `<script>` tag. As you learned earlier in this lesson, you use the `<script>` and `</script>` tags to enclose a script within the HTML document.

CAUTION

Remember to include only valid JavaScript statements between the starting and ending `<script>` tags. If the browser finds anything except valid JavaScript statements within the `<script>` tags, it will display a JavaScript error message. You should use the comment indicator (`//`) in front of any lines that are not JavaScript.

To begin creating the script, open your favorite text editor and type the beginning and ending `<script>` tags, as shown here:

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

In this script, you'll use JavaScript to determine the local and UTC times and then display them in the browser. Fortunately, all the hard parts, such as converting between date formats, are built in to the JavaScript interpreter; this is one of the reasons that displaying dates and times is a good starting place for beginners.

Storing Data in Variables

To begin the script, you will use a *variable* to store the current date. You will learn more about variables in Lesson 22, "Using JavaScript Variables, Strings, and Arrays," but for now just understand that a *variable* is a container that can hold a value—a number, some text, or, in this case, a date.

To start writing the script, add the following line after the first `<script>` tag, making sure to use the same combination of uppercase and lowercase letters in your version because JavaScript commands and variable names are case sensitive:

```
now = new Date();
```

This statement creates a variable called `now` and stores the current date and time in it. This statement and the others you will use in this script use JavaScript's built-in `Date` object, which enables you to conveniently handle dates and times. You'll learn more about working with dates in Lesson 22.

NOTE

Notice the semicolon at the end of the code snippet creating a variable called now. This semicolon tells the browser that it has reached the end of a statement. Semicolons are optional, but using them helps you avoid some common errors. We'll use them throughout these lessons for clarity.

Calculating the Results

Internally, JavaScript stores dates as the number of milliseconds since January 1, 1970. Fortunately, JavaScript includes a number of functions to convert dates and times in various ways, so you don't have to figure out how to convert milliseconds to days, dates, or times.

To continue your script, add the following two statements before the final `</script>` tag:

```
localtime = now.toString();
utctime = now.toGMTString();
```

These statements create two new variables: `localtime`, containing the current time and date in a nice readable format, and `utctime`, containing the UTC equivalent.

NOTE

The `localtime` and `utctime` variables store a piece of text, such as `January 1, 2001 12:00 PM`. In programming parlance, a piece of text is called a *string*.

Creating Output

You now have two variables—`localtime` and `utctime`—which contain the results you want from your script. Of course, these variables don't do you much good unless you can see them. JavaScript includes several ways to display information, and one of the simplest is by using the `document.write` statement.

The `document.write` statement displays a text string, a number, or anything else you throw at it. Because your JavaScript program will be used within a web page, the output will be displayed as part of the page. To display the result, add these statements before the final `</script>` tag:

```
document.write("<p><strong>Local time:</strong> " + localtime + "</p>");
document.write("<p><strong>UTC time:</strong> " + utctime + "</p>");
```

These statements tell the browser to add some text to the web page containing your script. The output will include some brief strings introducing the results and the contents of the `localtime` and `utctime` variables.

Notice the HTML elements, such as `<p>` and `<strong>`, within the quotation marks; because JavaScript's output appears within a web page, it needs to be formatted using HTML.

NOTE

Notice the plus signs (+) used between the text and variables in the `document.write()` code snippets. In this case, each plus sign tells the browser to combine the values into one string of text. If you use the plus sign between two numbers that aren't in quotes, they are added together.

Adding the Script to a Web Page

You should now have a complete script that calculates a result and displays it. Your script should match Listing 4.3.

LISTING 4.3 The Complete Date and Time Script

```
<script>
  now = new Date();
  localtime = now.toString();
  utctime = now.toGMTString();
  document.write("<p><strong>Local time:</strong> " + localtime + "</p>");
  document.write("<p><strong>UTC time:</strong> " + utctime + "</p>");
</script>
```

To use your script, you need to add it to an HTML document. If you use the general template you've seen in the lessons so far, you should end up with something like Listing 4.4.

LISTING 4.4 The Date and Time Script in an HTML Document

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Displaying Times and Dates</title>
  </head>
  <body>
    <h1>Current Date and Time</h1>
    <script>
     now = new Date();
     localtime = now.toString();
     utctime = now.toGMTString();
     document.write("<p><strong>Local time:</strong> "
          + localtime + "</p>");
     document.write("<p><strong>UTC time:</strong> " + utctime
          + "</p>");
    </script>
  </body>
</html>
```

Now that you have a complete HTML document, save it with an `.html` extension.

Testing the Script

To test your script, you simply need to load the HTML document you created in a web browser. If you typed the script correctly, your browser should display the result of the script, as shown in Figure 4.2. (Of course, your result won't be the same as mine, but it should be the same as the setting of your computer's clock.)

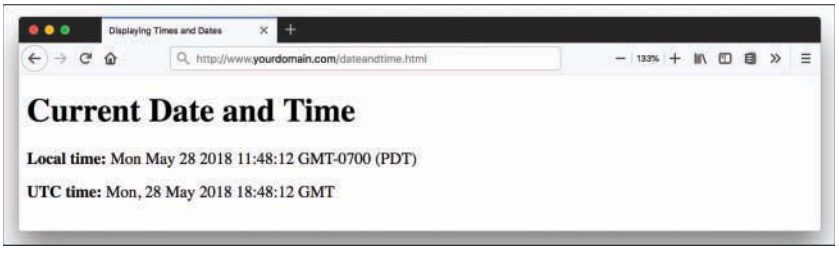

FIGURE 4.2
Using JavaScript to display the date and time.

NOTE

With Internet Explorer, depending on your security settings, the script might not execute, and your browser might display a security warning. In this case, follow your browser's instructions to allow your script to run. (This happens because the default security settings allow JavaScript in online documents but not in local files.)

Modifying the Script

Although the current script does indeed display the current date and time, its display isn't nearly as attractive as the clock on your wall or desk. To remedy that situation, you can use some additional JavaScript features and a bit of HTML to display a large clock.

To display a large clock, you need the hours, minutes, and seconds in separate variables. Once again, JavaScript has built-in functions to do most of the work:

```
hours = now.getHours();
mins = now.getMinutes();
secs = now.getSeconds();
```

These statements load the `hours`, `mins`, and `secs` variables with the components of the time using JavaScript's built-in date functions.

After the hours, minutes, and seconds are in separate variables, you can create `document.write` statements to display them:

```
document.write("<p><strong>");
document.write(hours + ":" + mins + ":" + secs);
document.write("</p></strong>");
```

The first statement displays an HTML <h2> header tag to display the clock as a second-level header element. The second statement displays the hours, mins, and secs variables, separated by colons, and the third adds the closing </h2> tag.

You can add the preceding statements to the original date and time script to add the large clock display. Listing 4.5 shows the complete modified version of the script.

LISTING 4.5 The Date and Time Script with a Large Clock Display

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Displaying Times and Dates</title>
  </head>
  <body>
    <h1>Current Date and Time</h1>
    <script>
     now = new Date();
     localtime = now.toString();
     utctime = now.toGMTString();
     document.write("<p><strong>Local time:</strong> "
          + localtime + "</p>");
     document.write("<p><strong>UTC time:</strong> "
          + utctime + "</p>");
     hours = now.getHours();
     mins = now.getMinutes();
     secs = now.getSeconds();
     document.write("<h2>");
     document.write(hours + ":" + mins + ":" + secs);
     document.write("</h2>");
    </script>
  </body>
</html>
```

Now that you have modified the script, save the HTML file and open the modified file in your browser. If you left the browser running, you can simply use the Reload button to load the new version of the script. Try it and verify that the same time is displayed in both the upper portion of the window and the new large clock. Figure 4.3 shows the results.

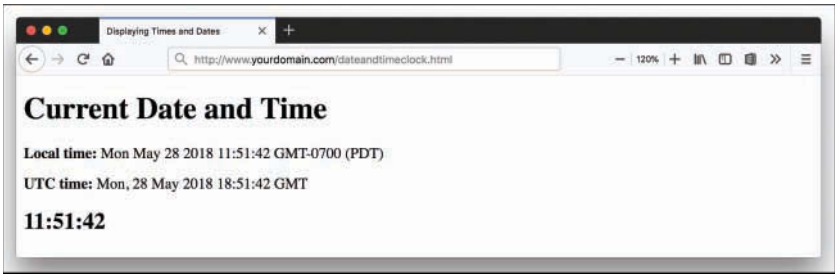

FIGURE 4.3
Displaying the modified date and time script.

NOTE

The time formatting produced by this script isn't perfect: Hours after noon are in 24-hour time, and there are no leading zeros, so 12:04 is displayed as 12:4. See Lesson 22 for solutions to these issues.

Dealing with JavaScript Errors

As you develop more complex JavaScript applications, you're going to run into errors from time to time. JavaScript errors are usually caused by mistyped JavaScript statements.

To see an example of a JavaScript error message, you can modify the statement you added in the preceding section. In this example, we use a common error: omitting one of the parentheses. Change the last `document.write` statement in Listing 4.5 to read

```
document.write("</h2>";
```

Save your HTML document again and load the document into the browser. Depending on the browser version you're using, one of two things will happen: Either an error message will be displayed, or the script will simply fail to execute.

If an error message is displayed, you're halfway to fixing the problem by adding the missing parenthesis. If no error was displayed, you should configure your browser to display error messages so that you can diagnose future problems:

- In Firefox, you can select Tools, Web Developer, Web Console. The console displays the error message you created in this example, as shown in Figure 4.4.
- In Chrome, from the options menu (three horizontal dots on the right side of the browser bar) select More Tools, Developer Tools. A console displays in the bottom of the browser window. Choose the console tab if it's not selected.

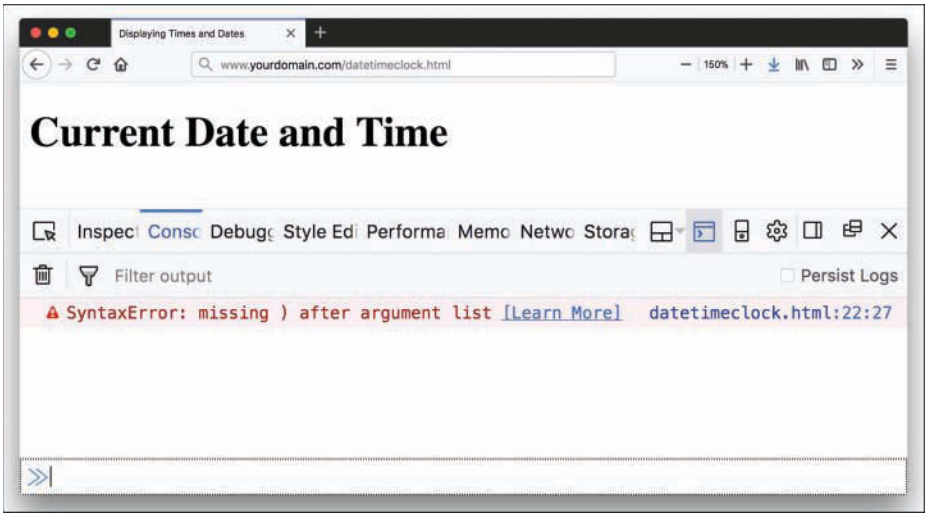

FIGURE 4.4
Showing an error in the JavaScript console in Firefox.

The error you get in this case, `SyntaxError: missing ) after argument list` , points to line 22. In this case, clicking the name of the document takes you directly to the highlighted line containing the error, as shown in Figure 4.5.

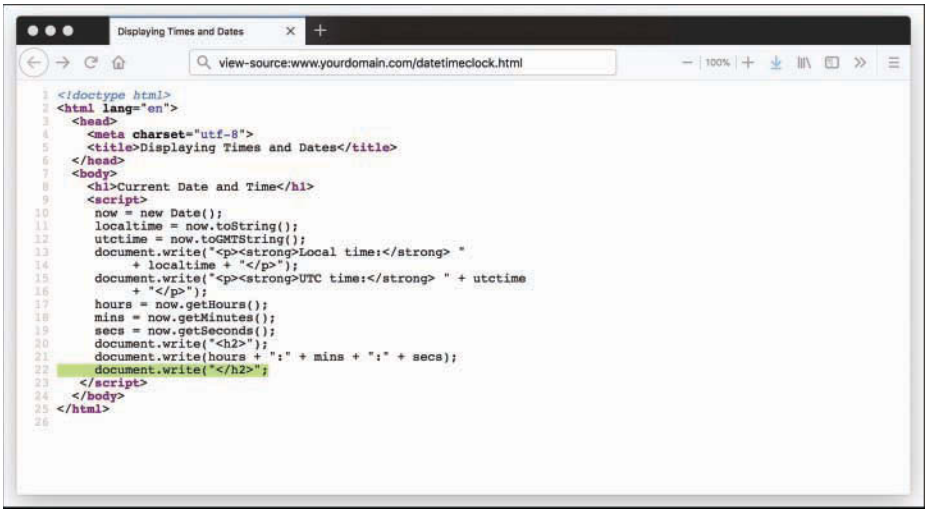

FIGURE 4.5
Firefox helpfully points out the offending line.

Most modern browsers contain JavaScript debugging tools such as the one you just witnessed. You'll learn more about this in the next lesson.

Summary

During this lesson, you've learned what web scripting is and what JavaScript is. You've also learned how to insert a script into an HTML document or refer to an external JavaScript file, what sorts of things JavaScript can do, and how JavaScript differs from other web languages. You also wrote a simple JavaScript program and tested it using a web browser. You also learned how to modify and test scripts, and you saw what happens when a JavaScript program runs into an error.

In the process of writing this script, you have used some of JavaScript's basic features: variables, the `document.write` statement, and functions for working with dates and times.

Now that you've learned a bit of JavaScript syntax, you're ready to continue on to learn all sorts of things about web development before settling in to writing interactive websites using client-side scripting.

Q&A

Q. Do I need to test my JavaScript on more than one browser?

A. In an ideal world, any script you write that follows the standards for JavaScript will work in all browsers, and 98% of the time (give or take) that's true in the real world. But browsers do have their quirks, and you should test your scripts in Chrome, Internet Explorer, and Firefox, as well as mobile devices running iOS and Android—at a minimum.

Q. If I plan to learn PHP, Ruby, or some other server-side programming language anyway, will I have any use for JavaScript?

A. Certainly. JavaScript is the ideal language for many parts of a web-based application, such as basic interactivity. Although PHP, Ruby, and other server-side languages have their uses, they can't interact directly with the user on the client side.

Q. When I try to run my script, the browser displays the actual script in the browser window instead of executing it. What did I do wrong?

A. This is most likely caused by one of three errors. First, you might be missing the beginning or ending `<script>` tags. Check them and, if you use the `type` attribute, verify that it reads `type="text/javascript"`. Second, your file might have been saved with a `.txt` extension, causing the browser to treat it as a text file. Rename it to have the extension `.htm` or `.html` to fix the problem. Third, make sure your browser supports JavaScript and ensure that it is not disabled in the preferences.

Q. Why are the `<strong>` and `<p>` tags allowed in the statements to print the time? I thought HTML tags aren't allowed within the `<script>` tags.

A. Because these tags are inside quotation marks, they are considered a valid part of the script. The script's output, including any HTML tags, is interpreted and displayed by the browser. You can use other HTML tags within quotation marks to add formatting, such as the `<h2>` tags you added for the large clock display.

Workshop

The workshop contains quiz questions and exercises to help you solidify your understanding of the material covered.

Quiz

1. When a user views a page containing a JavaScript program, which machine actually executes the script?

   a. The user's machine running a web browser

   b. The web server

   c. A central machine deep within Netscape's corporate offices

   d. A dedicated JavaScript server

2. What software do you use to create and edit JavaScript programs?

   a. A browser

   b. A text editor

   c. A pencil and a piece of paper

   d. A JavaScript editor

3. What are variables used for in JavaScript programs?

   a. Storing numbers, dates, or other values

   b. Varying randomly

   c. Causing high-school algebra flashbacks

   d. Changing the output of the script

4. What should appear at the very end of a JavaScript script embedded in an HTML file?

   a. The `<script>` tag

   b. The `</javascript>` tag

   c. The `END` statement

   d. The `</script>` tag

5. Which of these is not something you can do with JavaScript?

   a. Detect the features of the browser in use

   b. Modify part of a page without requiring a page refresh

   c. Write data to the remote server

   d. Interact with data from a remote server

6. Where can you place scripts?

   a. In the body of a page

   b. Within an HTML tag

   c. In a separate file

   d. All of the above

7. What do you use to include a separate script file in a page?

   a. `<link src="filename.js">`

   b. `<script src="filename.js"></script>`

   c. `<javascript src="filename.js">`

   d. `<include src="filename.js"></include>`

8. Which of these is an event handler?

   a. `<button>`

   b. `type="button"`

   c. `onclick="alert('You clicked the button.')"`

   d. `</button>`

9. What does the line `now = new Date();` do in JavaScript?

   a. It creates a variable called `now` and stores the current date in it.

   b. It creates a variable called `new` and stores the current date in it.

   c. It displays the current time.

   d. Nothing; it's not valid JavaScript.

10. Correct this line of JavaScript:

```
document.write("<p><strong>Dinner Time:</strong> " + localtime + "</p>);
```

   a.
```
document.write("<p><strong>Dinner Time:</strong> "
+ localtime + "</p>");
```

   b.
```
document.write("<p><strong>Dinner Time:</strong> "
+ localtime + "</p>');
```

   c.
```
document.write("<p><strong>Dinner Time:</strong> "
+ localtime + "</p>;
```

   d. The line is correct as written.

NOTE

Just a reminder for those of you reading these words in the print or e-book edition of this book: If you go to **www.informit.com/register** and register this book (using ISBN 9780672338083), you can receive free access to an online Web Edition that not only contains the complete text of this book but also features an interactive version of this quiz.

Answers

1. **a.** JavaScript programs execute in the web browser. (There is actually a server-side version of JavaScript, but that's another story.)
2. **b.** You can use any text editor to create scripts.
3. **a.** Variables are used to store numbers, dates, or other values.
4. **d.** Your script should end with the `</script>` tag.
5. **c.** JavaScript can never write data to the remote server due to security concerns.
6. **d.** JavaScript can be added in the body, a tag, or in a separate file.
7. **b.** Use the line `<script src="filename.js"></script>`.
8. **c.** The `onclick` attribute and its value are the click event handler.
9. **a.** It creates a variable called `now` and stores the current date in it.
10. **a.** The final quotation mark is missing.

Exercises

- Add a millisecond field to the large clock. You can use the `getMilliseconds` function, which works just like `getSeconds` but returns milliseconds.
- Modify the script to display the time, including milliseconds, twice. Notice whether any time passes between the two time displays when you load the page.

Index

Symbols

Numbers

A

B

C

D

E

F

G

H

J

K-L

M

N

O

P

Q

R

S

T

U

V

W

X-Y-Z