CHAPTER 4
Introducing JavaScript

What You'll Learn in This Chapter:

- 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 statements 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 your browser
- Moving scripts into separate files
- Basic syntax rules for avoiding JavaScript errors
- What JSON is and how it can be used
- Dealing with errors in scripts

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 a web scripting language such as JavaScript is one of the easiest ways to spice up a web page and to interact with users in new ways. In fact, using JavaScript is the next step in taking the static HTML you've learned about in the chapters so far and turning it into something dynamic.

The first part of this chapter introduces the concept of web scripting and the JavaScript language. As the chapter 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 one type of computer language: HTML. You use HTML tags to describe how you want your document formatted, and the browser obeys your commands and shows the formatted document to the user. But because HTML is a simple text markup language, it 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.

Whereas 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 to create interactive web pages.

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, an order form can 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 features supported by browsers and perform advanced functions only on browsers that support those features.
- 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

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.

How JavaScript Fits into a Web Page

Using the `<script>` tag, you can add a short script (in this case, just one line) to a web document, as shown in Listing 4.1. 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, which are described later in this chapter.

LISTING 4.1 A Simple HTML Document with a Simple Script

Click here to view code image

```
<!DOCTYPE html>

<html lang="en">
  <head>
    <title>A Spectacular Time!</title>
  </head>

  <body>
    <h1>It is a Spectacular Time!</h1>
    <p>What time is it, you ask?</p>
    <p>Well, indeed it is: <br>
    <script type="text/javascript">
    var currentTime = new Date();
    document.write(currentTime);
    </script>
    </p>
  </body>
</html>
```

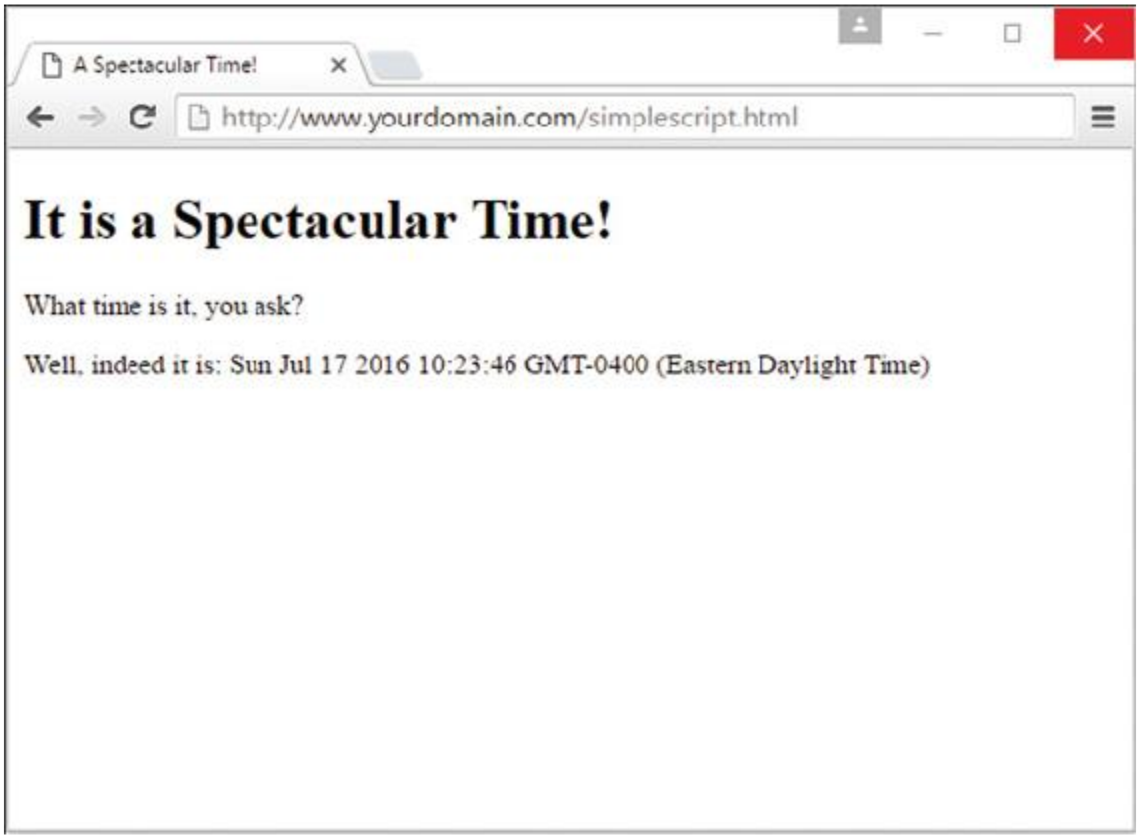
JavaScript's `document.write` method, which you'll learn more about later, sends output as part of the web document. In this case, it displays the current date and time, as shown in Figure 4.1.

In this example, we placed the script within the body of the HTML document. There are actually four places where you might place 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 because 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 Chapter 9, "Understanding JavaScript Event Handling."

- **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 Chapter 9.
- **In a separate file entirely**—JavaScript supports the use of files that can be included by specifying a file in the `<script>` tag. Although the `.js` extension is a convention, scripts can actually have any file extension, or none.



It is a Spectacular Time!

What time is it, you ask?

Well, indeed it is: Sun Jul 17 2016 10:23:46 GMT-0400 (Eastern Daylight Time)

FIGURE 4.1
Using `document.write` to display the current date.

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, like so—and don't forget to close the `<script>` tag using `</script>`:

Click here to view code image

```
<script type="text/javascript" src="filename.js"></script>
```

Because 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 using a text editor. It should contain one or more JavaScript statements, and only JavaScript—don't include `<script>` tags, other HTML tags, or HTML comments. Save the `.js` file in the same directory as the HTML documents that refer to it.

TIP

External JavaScript files have a distinct advantage: You can link to the same `.js` file from two or more HTML documents. Because the browser stores this file in its cache, the time it takes your web pages to display is reduced.

Using Basic 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

Click here to view code image

```
<!DOCTYPE html>

<html lang="en">
  <head>
    <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 this book, you'll learn more about JavaScript's event model and how to create simple and complex event handlers—in some cases, invoking PHP scripts for even greater dynamic experiences as you'll learn in Chapter 11, "AJAX: Getting Started with Remote Scripting".

Exploring JavaScript's Capabilities

If you've spent any time browsing the Web, you've undoubtedly seen lots of examples of JavaScript, even if you were not aware that JavaScript was powering your interactions. Here are some brief descriptions of typical applications for JavaScript.

Validating Forms

Form validation is a common use of JavaScript, although the form validation features of HTML5 have stolen a lot of JavaScript's thunder here. 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.

Special Effects

One of the earliest, and admittedly 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. Additionally, some developers use HTML5, CSS3, and JavaScript in tandem to create fully functioning interactive games.

Remote Scripting (AJAX)

For a long time, one of the greatest limitations 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 you could not use JavaScript to look up the user's location in a database based on the number.

However, now your scripts can get data from a server without loading a page or sending data back to be saved. These features are collectively known as AJAX (Asynchronous JavaScript and XML), or *remote scripting*. You'll learn how to develop AJAX scripts in Chapter 11.

You've seen AJAX in action if you've used Google's Gmail mail application, Facebook, or any online news site that allows you to comment on stories, vote for favorites, or participate in a poll (among many other things). All of these use remote scripting to present you with a dynamic user interface that interacts with a server in the background.

Basic JavaScript Language Concepts

As you build on your knowledge, there are a few foundational concepts you should know that form the basic building blocks of JavaScript.

Statements

Statements are the basic units of programs, be it a JavaScript or PHP program (or any other language). A statement is a section of code that performs a single action. For example, the following four statements create a new `Date` object and then assign the values for the current hour, minutes, and seconds into variables called `hours`, `mins`, and `secs`, respectively. You can then use these variables elsewhere in your JavaScript code.

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

Although a statement is typically a single line of JavaScript, this is not a rule—it's possible (and fairly common) to break a statement across multiple lines, or to include more than one statement in a single line.

A semicolon marks the end of a statement, but you can also omit the semicolon if you start a new line after the statement—if that is your coding style. In other words, these are three valid JavaScript statements:

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

However, if you combine statements into a single line, you must use semicolons to separate them. For example, the following line is valid:

Click here to view code image

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

But this line is invalid:

Click here to view code image

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

Again, your style is always up to you but I personally recommend always using semicolons to end statements.

Combining Tasks with Functions

Functions are groups of JavaScript statements that are treated as a single unit—this is a term you'll use in PHP as well. A statement that uses a function is referred to as a *function call*. For example, you might create a function called `alertMe`, which produces an alert when called, like so:

Click here to view code image

```
function alertMe() {
   alert("I am alerting you!");
}
```

When this function is called, a JavaScript alert pops up and the text `I am alerting you!` is displayed.

Functions can take arguments—the expression inside the parentheses—to tell them what to do. Additionally, a function can return a value to a waiting variable. For example, the following function call prompts the user for a response and stores it in the `text` variable:

Click here to view code image

```
text = prompt("Enter some text.")
```

Creating your own functions is useful for two main reasons: First, you can separate logical portions of your script to make it easier to understand. Second, and more important, you can use the function several times or with different data to avoid repeating script statements.

NOTE

Entire chapters of this book are devoted to learning how to create and use functions both in JavaScript and in PHP.

Variables

Variables are containers that can store a number, a string of text, or another value. For example, the following statement creates a variable called `food` and assigns it the value `cheese`:

```
var food = "cheese";
```

JavaScript variables can contain numbers, text strings, and other values. You'll learn more about variables in much greater detail in Chapter 7, "JavaScript Fundamentals: Variables, Strings, and Arrays," and in the context of PHP in Chapter 12, "PHP Fundamentals: Variables, Strings, and Arrays."

Objects

JavaScript also supports *objects*. Like variables, objects can store data—but they can store two or more pieces of data at once. As you'll learn throughout the JavaScript-specific chapters of this book, using built-in objects and their methods is fundamental to JavaScript—it's one of the ways the language works, by providing a predetermined set of actions you can perform. For example, the `document.write` functionality you saw earlier in this chapter is actually a situation in which you use the `write` method of the `document` object to output text to the browser for eventual rendering.

The data stored in an object is called a *property* of the object. For example, you could use objects to store information about people in an address book. The properties of each person object might include a name, an address, and a telephone number.

You'll want to become intimately familiar with object-related syntax, because you will see objects quite a lot, even if you don't build your own. You'll definitely find yourself using built-in objects, and objects will very likely form a large part of any JavaScript libraries you import for use. JavaScript uses periods to separate object names and property names. For example, for a person object called Bob, the properties might include `Bob.address` and `Bob.phone`.

Objects can also include *methods*. These are functions that work with the object's data. For example, our person object for the address book might include a `display()` method to display the person's information. In JavaScript terminology, the statement `Bob.display()` would display Bob's details.

Don’t worry if this sounds confusing—you’ll be exploring objects in much more detail later in this book both in the context of learning JavaScript fundamentals and PHP fundamentals. For now, you just need to know the basics, which are that JavaScript supports three kinds of objects:

- *Built-in objects* are built in to the JavaScript language. You’ve already encountered one of these: `Date`. Other built-in objects include `Array`, `String`, `Math`, `Boolean`, `Number`, and `RegExp`.
- *DOM (Document Object Model) objects* represent various components of the browser and the current HTML document. For example, the `alert()` function you used earlier in this chapter is actually a method of the `window` object.
- *Custom objects* are objects you create yourself. For example, you could create a `Person` object, as mentioned earlier in this section.

Conditionals

Although you can use event handlers to notify your script (and potentially the user) when something happens, you might need to check certain conditions yourself as your script runs. For example, you might want to validate on your own that a user entered a valid email address in a web form.

JavaScript supports *conditional statements*, which enable you to answer questions like this. A typical conditional uses the `if` keyword, as in this example:

Click here to view code image

```
if (count == 1) {
   alert("The countdown has reached 1.");
}
```

This compares the variable count with the constant `1` and displays an alert message to the user if they are the same. It is quite likely you will use one or more conditional statements like this in most of your scripts, and more space is devoted to this concept in Chapter 8, “JavaScript Fundamentals: Functions, Objects, and Flow Control.”

Loops

Another useful feature of JavaScript—and most other programming languages—is the capability to create *loops*, or groups of statements that repeat a certain number of times. For example, these statements display the same alert 10 times, greatly annoying the user but showing how loops work:

Click here to view code image

```
for (i=1; i<=10; i++) {
   alert("Yes, it's yet another alert!");
}
```

The `for` statement is one of several statements JavaScript uses for loops. This is the sort of thing computers are supposed to be good at—performing repetitive tasks. You will use loops in many of your scripts, in much more useful ways than this example, as you'll see in Chapter 8.

Event Handlers

As mentioned previously, not all scripts are located within `<script>` tags. You can also use scripts as *event handlers*. Although this might sound like a complex programming term, it actually means exactly what it says: Event handlers are scripts that handle events. You learned a little bit about events already, but not to the extent you'll read about now or learn in Chapter 9, "Understanding JavaScript Event Handling."

In real life, an event is something that happens to you. For example, the things you write on your calendar are scheduled events, such as "Dentist appointment" and "Fred's birthday." You also encounter unscheduled events in your life, such as a traffic ticket and an unexpected visit from relatives.

Whether events are scheduled or unscheduled, you probably have normal ways of handling them. Your event handlers might include things such as *When Fred's birthday arrives, send him a present* and *When relatives visit unexpectedly, turn off the lights and pretend nobody is home*.

Event handlers in JavaScript are similar: They tell the browser what to do when a certain event occurs. The events JavaScript deals with aren't as exciting as the ones you deal with—they include such events as *When the* *mouse button is pressed* and *When this page is finished loading*. Nevertheless, they’re a very useful part of JavaScript’s environment.

Many events (such as mouse clicks, which you’ve seen previously) are caused by the user. Rather than doing things in a set order, your script can respond to the user’s actions. Other events don’t involve the user directly—for example, an event can be triggered when an HTML document finishes loading.

Each event handler is associated with a particular browser object, and you can specify the event handler in the tag that defines the object. For example, images and text links have an event, `onmouseover`, that happens when the mouse pointer moves over the object. Here is a typical HTML image tag with an event handler:

Click here to view code image

```
<img src="button.gif" onmouseover="highlight();">
```

You specify the event handler as an attribute within the HTML tag and include the JavaScript statement to handle the event within the quotation marks. This is an ideal use for functions because function names are short and to the point and can refer to a whole series of statements.

▼ TRY IT YOURSELF

Using an Event Handler

Here’s a simple example of an event handler that will give you some practice setting up an event and working with JavaScript without using `<script>` tags. Listing 4.3 shows an HTML document that includes a simple event handler.

LISTING 4.3 An HTML Document with a Simple Event Handler

Click here to view code image

```
<!DOCTYPE html>
```

```
<html lang="en">
  <head>
    <title>Simple Event Handler Example</title>
  </head>

  <body>
    <h1>Simple Event Handler Example</h1>
     <p><a href="http://www.google.com/"
           onclick="alert('A-ha! An Event!');">Go to Google</a>
     </p>
  </body>
</html>
```

The event handler is defined with the following `onclick` attribute within the `<a>` tag that defines a link:

Click here to view code image

```
onclick="alert('A-ha! An Event!');"
```

This event handler uses the DOM's built-in `alert` method of the `window` object to display a message when you click the link; after you click OK to dismiss the alert, your browser will continue on to the URL. In more complex scripts, you will usually define your own functions to act as event handlers.

You'll use other event handlers throughout this book, leading up to a more comprehensive lesson in Chapter 9.

After you click the OK button to dismiss the alert, the browser follows the link defined in the `<a>` tag. Your event handler could also stop the browser from following the link, as you will learn in Chapter 9.

Which Script Runs First?

You are not limited to a single script within a web document: one or more sets of `<script>` tags, external JavaScript files, and any number of event handlers can be used within a single document. With all of these scripts, you might wonder how the browser knows which to execute first. Fortunately, this is done in a logical fashion:

- Sets of `<script>` tags within the `<head>` element of an HTML document are handled first, whether they include embedded code or refer to a JavaScript file. Because scripts in the `<head>` element will not create output in the web page, it's a good place to define functions for use later.
- Sets of `<script>` tags within the `<body>` section of the HTML document are executed after those in the `<head>` section, while the web page loads and displays. If there are two or more scripts in the body, they are executed in order.
- Event handlers are executed when their events happen. For example, the `onload` event handler is executed when the body of a web page loads. Because the `<head>` section is loaded before any events, you can define functions there and use them in event handlers.

JavaScript Syntax Rules

JavaScript is a simple language, but you do need to be careful to use its *syntax*—the rules that define how you use the language—correctly. The rest of this book covers many aspects of JavaScript syntax, but there are a few basic rules you can begin to keep in mind now, throughout the chapters in this book, and then when you are working on your own.

Case Sensitivity

Almost everything in JavaScript is *case sensitive*: You cannot use lowercase and capital letters interchangeably. Here are a few general rules:

- JavaScript keywords, such as `for` and `if`, are always lowercase.
- Built-in objects such as `Math` and `Date` are capitalized.
- DOM object names are usually lowercase, but their methods are often a combination of capitals and lowercase. Usually capitals are used for all but the first word, as in `setAttribute` and `getElementById`.

When in doubt, follow the exact case used in this book or another JavaScript reference. If you use the wrong case, the browser will usually display an error message.

Variable, Object, and Function Names

When you define your own variables, objects, or functions, you can choose their names. Names can include uppercase letters, lowercase letters, numbers, and the underscore (_) character. Names must begin with a letter or an underscore.

You can choose whether to use capitals or lowercase in your variable names, but remember that JavaScript is case sensitive: `score`, `Score`, and `SCORE` would be considered three different variables. Be sure to use the same name each time you refer to a variable.

Reserved Words

One more rule for variable names: They must not be *reserved words*. These include the words that make up the JavaScript language, such as `if` and `for`, DOM object names such as `window` and `document`, and built-in object names such as `Math` and `Date`.

A list of JavaScript reserved words can be found at https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Reserved_Words.

Spacing

Blank space (known as *whitespace* by programmers) is mostly ignored by JavaScript. You can usually include spaces and tabs within a line, or blank lines, without causing an error, although if your statements are not clearly terminated or bracketed you may get into a sticky situation. Overall, whitespace often makes the script more readable, so do not hesitate to use it.

Using Comments

JavaScript *comments* enable you to include documentation within your script. Brief documentation is useful if someone else needs to understand the script, or even if you try to understand it after returning to your code after a long break. To include comments in a JavaScript program, begin a line with two slashes, as in this example:

```
// this is a comment.
```

You can also begin a comment with two slashes in the middle of a line, which is useful for documenting a script. In this case, everything on the line after the slashes is treated as a comment and ignored by the JavaScript. For example, the following line is a valid JavaScript statement followed by a comment explaining what is going on in the code:

Click here to view code image

```
a = a + 1; // add 1 to the value of the variable a
```

JavaScript also supports C-style comments, which begin with `/*` and end with `*/`. These comments can extend across more than one line, as the following example demonstrates:

Click here to view code image

```
/* This script includes a variety
of features, including this comment. */
```

Because JavaScript statements within a comment are ignored, this type of comment is often used for *commenting out* sections of code. If you have some lines of JavaScript that you want to temporarily take out of the picture while you debug a script, you can add `/*` at the beginning of the section and `*/` at the end.

Best Practices for JavaScript

Now that you've learned some of the very basic rules for writing valid JavaScript, it's also a good idea to follow a few *best practices*. The following practices are not required, but you'll save yourself and others some headaches if you begin to integrate them into your development process:

- *Use comments liberally*. These make your code easier for others to understand, and also easier for you to understand when you edit it later. Comments are also useful for marking the major divisions of a script.
- *Use a semicolon at the end of each statement, and use only one statement per line*. Although you learned in this chapter that semicolons are not necessary to end a statement (if you use a new line), using semicolons and only one statement per line will make your scripts easier to read and also easier to debug.
- *Use separate JavaScript files whenever possible*. Separating large chunks of JavaScript makes debugging easier, and also encourages you to write modular scripts that can be reused.
- *Avoid being browser-specific*. As you learn more about JavaScript, you'll learn some features that work in only one browser. Avoid them unless absolutely necessary, and always test your code in more than one browser.
- *Keep JavaScript optional*. Don't use JavaScript to perform an essential function on your site—for example, the primary navigation links. Whenever possible, users without JavaScript should be able to use your site, although it might not be quite as attractive or convenient. This strategy is known as *progressive enhancement*.

There are many more best practices involving more advanced aspects of JavaScript. You'll learn about them not only as you progress through the chapters, but also over time and as you collaborate with others on web development projects.

Understanding JSON

Although JSON, or *JavaScript Object Notation*, is not a part of the core JavaScript language, it is in fact a common way to structure and store information either used by or created by JavaScript-based functionality on the client side. Now is a good time to familiarize yourself with JSON (pronounced “Jason”) and some of its uses.

JSON-encoded data is expressed as a sequence of parameter and value pairs, with each pair using a colon to separate parameter from value. These “parameter”:“value” pairs are themselves separated by commas, as shown here:

Click here to view code image

```
"param1":"value1", "param2":"value2", "param3":"value3"
```

Finally, the whole sequence is enclosed between curly braces to form a JSON object; the following example creates a variable called `yourJSONObject`:

```
var yourJSONObject = {
   "param1":"value1",
   "param2":"value2",
   "param3":"value3"
}
```

Note that there is no comma after the final parameter—doing so would be a syntax error since no additional parameter follows.

JSON objects can have properties accessed directly using the usual dot notation, such as this:

Click here to view code image

```
alert(yourJSONObject.param1); // alerts 'value1'
```

More generally, though, JSON is a general-purpose syntax for exchanging data in a string format. It is then easy to convert the JSON object into a string by a process known as serialization; serialized data is convenient for storage or transmission around networks. You’ll see some uses of serialized JSON objects as this book progresses.

One of the most common uses of JSON these days is as a data interchange format used by application programming interfaces (APIs) and other data feeds consumed by a front-end application that uses JavaScript to parse this data. This increased use of JSON in place of other data formats such as XML has come about because JSON is

- Easy to read for both people and computers
- Simple in concept, with a JSON object being nothing more than a series of “parameter”:“value” pairs enclosed by curly braces
- Largely self-describing
- Fast to create and parse
- No special interpreters or other additional packages are necessary

Using the JavaScript Console to Debug JavaScript

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, modify the statement you added in the preceding section. We’ll use a common error: omitting one of the parentheses. Change the last `document.write` statement in Listing 4.1 to read as follows:

```
document.write(currentTime;
```

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.

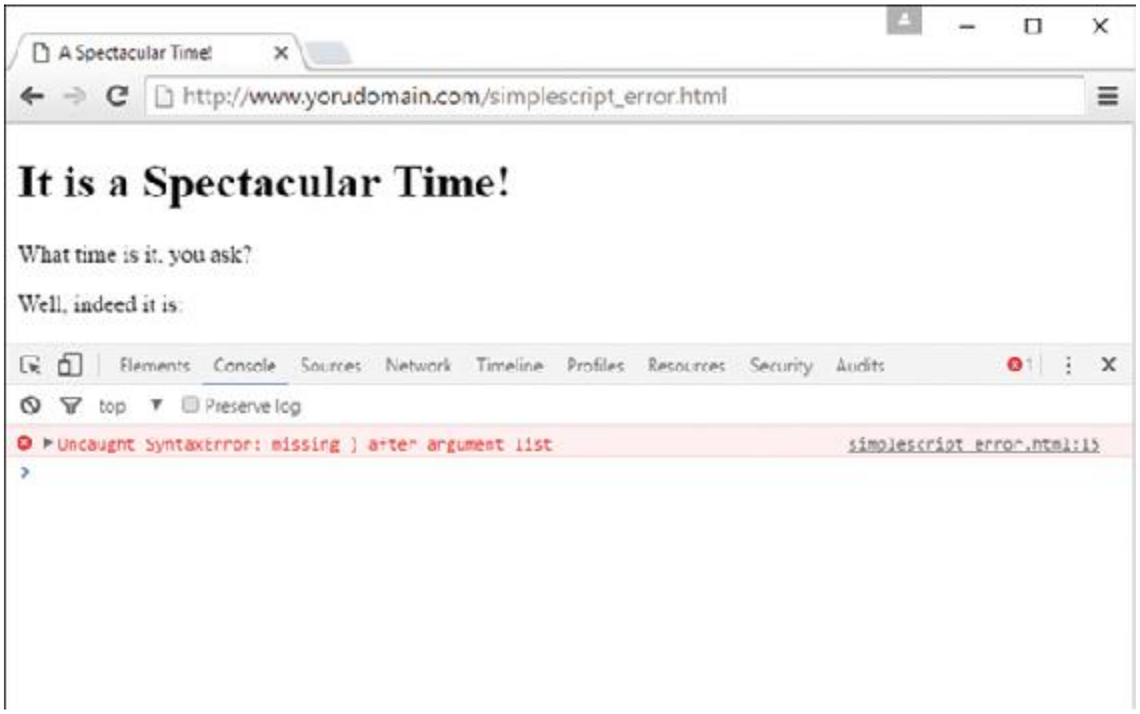
If an error message is displayed, you’re halfway to fixing the problem by adding the missing parenthesis. If no error was displayed, you should configure your browser to display error messages so that you can diagnose future problems:

- In Firefox, you can select the Developer option, then Browser Console from the main menu.

- In Chrome, select More Tools, Developer Tools from the main menu. A console displays in the bottom of the browser window. The console is shown in Figure 4.2, displaying the error message you created in this example.
- In Microsoft Edge, select F12 Developer Tools from the main menu.

The error we get in this case is `Uncaught SyntaxError` and it points to line 15. In this case, clicking the name of the script takes you directly to the highlighted line containing the error, as shown in Figure 4.3.

Most modern browsers contain JavaScript debugging tools such as the one you just witnessed. These tools will be incredibly useful to you as you begin your journey in web application development.



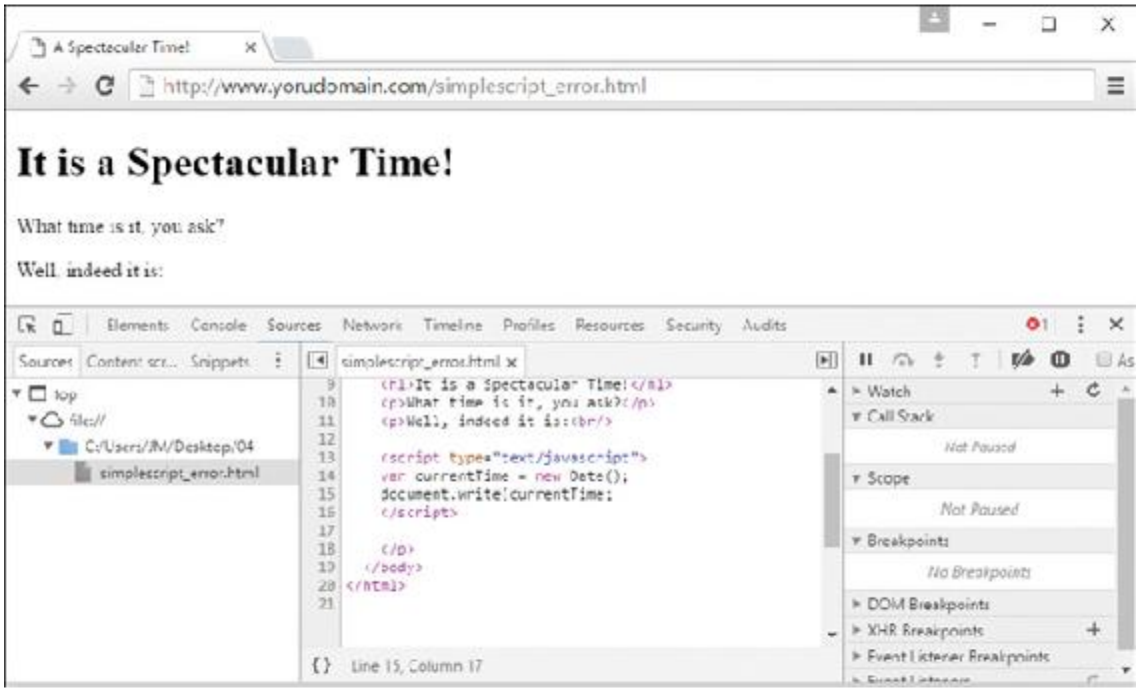
It is a Spectacular Time!

What time is it, you ask?

Well, indeed it is.

```
Uncaught SyntaxError: missing ) after argument list          simplescript_error.html:15
```

FIGURE 4.2
Showing an error in the JavaScript console in Chrome.



It is a Spectacular Time!

What time is it, you ask?

Well, indeed it is:

```
    <h1>It is a Spectacular Time!</h1>
    <p>What time is it, you ask?</p>
    <p>Well, indeed it is:<br/>

    <script type="text/javascript">
    var currentTime = new Date();
    document.write(currentTime;
    </script>

    </p>
  </body>
</html>
```

FIGURE 4.3
Chrome helpfully points out the offending line.

Summary

In this chapter, you learned what web scripting is and what JavaScript is. You 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.

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. You were also introduced to the concept of JavaScript functions, objects, event handlers, conditions, and loops.

Additionally, you learned how to use JavaScript comments to make your script easier to read, and we looked at a simple example of an event handler. Finally, you were introduced to JSON, a data interchange format that is commonly used by JavaScript-based applications, and what happens when a JavaScript program runs into an error (and how to debug it).

Now that you’ve learned a bit of JavaScript syntax, you’re ready to build on that in future chapters. But before that, take a moment in the next chapter to learn a similar set of basic information about a server-side scripting language: PHP.

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 at a minimum.

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 then also verify that the first reads `<script type="text/javascript">`. Finally, your file might have been saved with a `.txt` extension, causing the browser to treat it as a text file. Rename it to `.htm` or `.html` to fix the problem.

Q. I’ve heard the term *object-oriented* applied to languages such as C++ and Java. If JavaScript supports objects, is it an object-oriented language?

A. Yes, although it might not fit some people’s strict definitions. JavaScript objects do not support all the features that languages such as C++ and Java support, although the latest versions of JavaScript have added more object-oriented features.

Workshop

The Workshop contains quiz questions and exercises to help you solidify your understanding of the material covered. Try to answer all questions before looking at the "Answers" section that follows.

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

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

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

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

   A. The `<script type="text/javascript">` tag

   B. The `</script>` tag

   C. The `END` statement

5. Which of the following is executed first by a browser?

   A. A script in the `<head>` section

   B. A script in the `<body>` section

   C. An event handler for a button

Answers

1. **A.** JavaScript programs execute on the web browser. (There is actually a server-side version of JavaScript, but that's another story.)
2. **B.** Any text editor can be used to create scripts. You can also use a word processor if you're careful to save the document as a text file with the `.html` or `.htm` extension.
3. **A.** Variables are used to store numbers, dates, or other values.
4. **B.** Your script should end with the `</script>` tag.
5. **A.** Scripts defined in the `<head>` section of an HTML document are executed first by the browser.

Exercises

- Examine the simple time display script you created and comment the code to explain everything that is happening all along the way. Verify that the script still runs properly.
- Use the techniques you learned in this chapter to create a script that displays the current date and time in an alert box when the user clicks a link.