HOUR 3

Getting Started with JavaScript Programming

What You'll Learn in This Hour:

- Organizing scripts using functions
- What objects are and how JavaScript uses them
- How JavaScript can respond to events
- An introduction to conditional statements and loops
- How browsers execute scripts in the proper order
- Syntax rules for avoiding JavaScript errors
- Adding comments to document your JavaScript code

You've reached the halfway point of Part I of this book. In the first couple of hours, you've learned what JavaScript is, learned the variety of things JavaScript can do, and created a simple script.

In this hour, you'll learn a few basic concepts and script components that you'll use in just about every script you write. This will prepare you for the remaining hours of this book, in which you'll explore specific JavaScript functions and features.

Basic Concepts

There are a few basic concepts and terms you'll run into throughout this book. In the following sections, you'll learn about the basic building blocks of JavaScript.

Statements

Statements are the basic units of a JavaScript program. A statement is a section of code that performs a single action. For example, the following three statements are from the date and time example in Hour 2, "Creating Simple Scripts":

```
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 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. You can also omit the semicolon if you start a new line after the statement. If you combine statements into a single line, you must use semicolons to separate them.

Combining Tasks with Functions

In the basic scripts you've examined so far, you've seen some JavaScript statements that have a section in parentheses, like this:

```
document.write("Testing.");
```

This is an example of a *function*. Functions provide a simple way to handle a task, such as adding output to a web page. JavaScript includes a wide variety of built-in functions, which you will learn about throughout this book. A statement that uses a function, as in the preceding example, is referred to as a *function call*.

Functions take parameters (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:

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

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

By the Way

You will learn how to define, call, and return values from your own functions in Hour 6, "Using Functions and Objects."

Variables

In Hour 2, you learned that variables are containers that can store a number, a string of text, or another value. For example, the following statement creates a variable called `fred` and assigns it the value 27:

```
var fred = 27;
```

JavaScript variables can contain numbers, text strings, and other values. You'll learn more about them in Hour 5, "Using Variables, Strings, and Arrays."

Understanding Objects

JavaScript also supports *objects*. Like variables, objects can store data—but they can store two or more pieces of data at once.

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

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.

By the Way

The `document.write` function we discussed earlier this hour is actually the `write` method of the `document` object. You will learn more about this object in Hour 4, "Working with the Document Object Model (DOM)."

Don't worry if this sounds confusing—you'll be exploring objects in much more detail later in this book. For now, you just need to know the basics. JavaScript supports three kinds of objects:

- *Built-in objects* are built in to the JavaScript language. You've already encountered one of these, `Date`, in Hour 2. Other built-in objects include `Array` and `String`, which you'll explore in Hour 5, and `Math`, which is explained in Hour 8, "Using Built-in Functions and Libraries."
- *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 hour is actually a method of the `window` object. You'll explore these in more detail in Hour 4.

- *Custom objects* are objects you create yourself. For example, you could create a `person` object, as in the examples in this section. You'll learn to use custom objects in Hour 6.

Conditionals

Although event handlers notify your script when something happens, you might want to check certain conditions yourself. For example, did the user enter a valid email address?

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

```
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. You will use conditional statements like this in most of your scripts.

By the Way

You'll learn more about conditionals in Hour 7, "Controlling Flow with Conditions and Loops."

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:

```
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.

By the Way

Loops are covered in detail in Hour 7.

Event Handlers

As mentioned in Hour 1, "Understanding JavaScript," 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.

In real life, an event is something that happens to you. For example, the things you write on your calendar are events: "Dentist appointment" or "Fred's birthday." You also encounter unscheduled events in your life: for example, a traffic ticket, an IRS audit, or 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* or *When relatives visit unexpectedly, turn out 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 clicks* and *When this page is finished loading*. Nevertheless, they're a very useful part of JavaScript.

Many JavaScript events (such as mouse clicks) 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 is 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:

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

You specify the event handler as an attribute to 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.

See the Try It Yourself section at the end of this hour for a complete example of an event handler within an HTML document.

By the Way

You can also define event handlers within JavaScript without using HTML attributes. You'll learn this technique, and more about event handlers, in Hour 9, "Responding to Events."

Which Script Runs First?

You can actually have several scripts within a web document: one or more sets of `<script>` tags, external JavaScript files, and any number of event handlers. 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>` section of an HTML document are handled first, whether they include embedded code or refer to a JavaScript file. Because these scripts cannot 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 is more than one script 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 should understand to avoid errors.

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 `toLowerCase` 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 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 complete list of reserved words is included in Appendix D, "JavaScript Quick Reference."

Spacing

Blank space (known as *whitespace* by programmers) is ignored by JavaScript. You can include spaces and tabs within a line, or blank lines, without causing an error. Blank space often makes the script more readable.

Using Comments

JavaScript *comments* enable you to include documentation within your script. This will be useful if someone else tries to understand the script, or even if you try to understand it 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 browser. For example,

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

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

Because JavaScript statements within a comment are ignored, C-style comments are 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.

By the Way

Because these comments are part of JavaScript syntax, they are only valid inside `<script>` tags or within an external JavaScript file.

Best Practices for JavaScript

You should now be familiar with the basic rules for writing valid JavaScript. Along with following the rules, it's also a good idea to follow *best practices*. The following practices may not be required, but you'll save yourself and others some headaches if you follow them.

- **Use comments liberally**—These make your code easier for others to understand, and also easier for you to understand when you edit them later. They are also useful for marking the major divisions of a script.
- **Use a semicolon at the end of each statement, and only use one statement per line**—This will make your scripts easier to debug.
- **Use separate JavaScript files whenever possible**—This separates JavaScript from HTML and 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 only work in 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 may 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. These are covered in detail in Hour 15, "Unobtrusive Scripting."

Try It Yourself ▼

Using an Event Handler

To conclude this hour, here's a simple example of an event handler. This will demonstrate how you set up an event, which you'll use throughout this book, and how JavaScript works without `<script>` tags. Listing 3.1 shows an HTML document that includes a simple event handler.

LISTING 3.1 An HTML Document with a Simple Event Handler

```
<html>
<head>
<title>Event Handler Example</title>
</head>
<body>
<h1>Event Handler Example</h1>
<p>
<a href="http://www.jsworkshop.com/"
onClick="alert('Aha! An Event!');">Click this link</a>
to test an event handler.
</p>
</body>
</html>
```

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

```
onClick="alert('Aha! An Event!');"
```

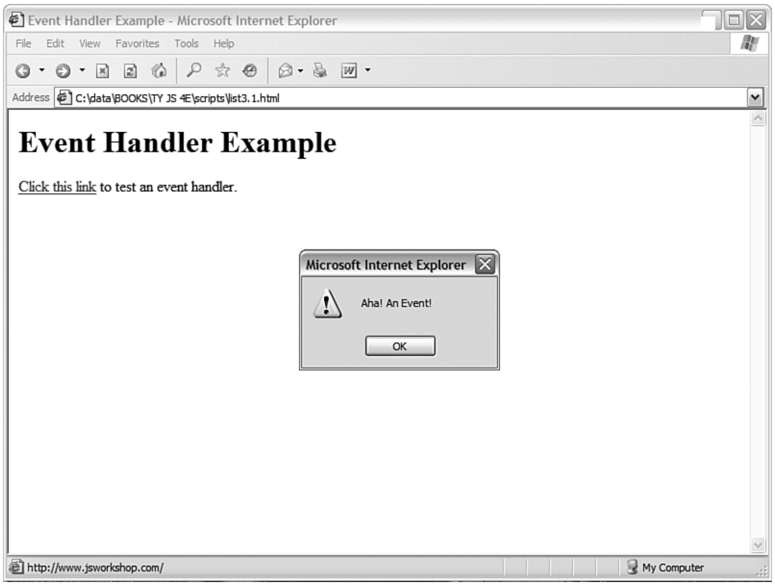
This event handler uses the built-in `alert()` function to display a message when you click on the link. In more complex scripts, you will usually define your own function to act as an event handler. Figure 3.1 shows this example in action.

You'll use other event handlers similar to this in the next hour, and events will be covered in more detail in Hour 9.

Did you Know?

Notice that after you click the OK button on 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 described in Hour 9.

FIGURE 3.1
The browser displays an alert when you click the link.


Event Handler Example - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address C:\data\BOOKS\TY JS 4E\scripts\list3.1.html
Event Handler Example
Click this link to test an event handler.
Microsoft Internet Explorer
Aha! An Event!
OK
http://www.jsworkshop.com/
My Computer

Summary

During this hour, you've been introduced to several components of JavaScript programming and syntax: functions, objects, event handlers, conditions, and loops. You also learned how to use JavaScript comments to make your script easier to read, and looked at a simple example of an event handler.

In the next hour, you'll look at the Document Object Model (DOM) and learn how you can use the objects within the DOM to work with web pages and interact with users.

Q&A

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 of the features that languages such as C++ and Java support, although the latest versions of JavaScript have added more object-oriented features.

Q. ***Having several scripts that execute at different times seems confusing. Why would I want to use event handlers?***

A. Event handlers are the ideal way (and in JavaScript, the only way) to handle gadgets within the web page, such as buttons, check boxes, and text fields. It's actually more convenient to handle them this way. Rather than writing a script that sits and waits for a button to be pushed, you can simply create an event handler and let the browser do the waiting for you.

Q. ***Some examples in other books suggest enclosing scripts in HTML comments (`<!--` and `-->`) to hide the script from older browsers. Is this necessary?***

A. This technique was only necessary for supporting very old browsers, such as Netscape 2.0. I no longer recommend this because all modern browsers handle JavaScript correctly. If you are still concerned about non-JavaScript browsers, the best way to hide your script is to use an external JavaScript file, as described in Hour 2.

Quiz Questions

Test your knowledge of JavaScript by answering the following questions:

1. A script that executes when the user clicks the mouse button is an example of what?
   - **a.** An object
   - **b.** An event handler
   - **c.** An impossibility
2. Which of the following are capabilities of functions in JavaScript?
   - **a.** Accept parameters
   - **b.** Return a value
   - **c.** Both of the above
3. 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

Quiz Answers

1. b. A script that executes when the user clicks the mouse button is an event handler.
2. c. Functions can accept both parameters and return values.
3. a. Scripts defined in the `<head>` section of an HTML document are executed first by the browser.

Exercises

To further explore the JavaScript features you learned about in this hour, you can perform the following exercises:

- Examine the Date and Time script you created in Hour 2 and find any examples of functions and objects being used.
- Add JavaScript comments to the Date and Time script to make it more clear what each line does. Verify that the script still runs properly.