HOUR 3

Understanding JavaScript

DHTML isn't a new Web language—it's just a set of objects that allows you to use a scripting language (such as JavaScript) to manipulate the page you set up in standard HTML.

In this hour, you'll review the essentials of the JavaScript language. If you're already a JavaScript expert, feel free to skim this chapter, but you should have a solid grasp of JavaScript to understand the DHTML examples in the rest of this book.

The following topics will be covered in this hour:

- The basics of JavaScript syntax
- How to avoid problems with older browsers
- How variables, strings, and arrays work in JavaScript
- Using JavaScript objects to store data and functions
- Using Conditional statements in JavaScript
- Using loops to repeat JavaScript statements
- Creating and using functions in JavaScript

JavaScript Basics

JavaScript was the first of the client-side Web scripting languages. It first appeared in 1995 in Netscape 2.0. You can use JavaScript directly within a Web page without using any special tools to create or compile scripts, and it works on most of today's browsers.

Despite the name, JavaScript has little to do with the Java language. Some of the commands are similar, but it's a different language with a different purpose.

This hour will only give you a quick review of the essentials of JavaScript. See Appendix A, "Other JavaScript and DHTML Resources," for a list of Web sites and books with further information.

Beginning and Ending Scripts

You can place JavaScript commands anywhere in an HTML document. To begin and end the script, you use the `<script>` tag. This tells the browser to interpret the commands between the tags as JavaScript rather than HTML.

The `<script>` tag specifies the language to be used. Here is a simple example of a JavaScript command with beginning and ending `<script>` tags:

```
<script language="JavaScript">
document.write("Hello World");
</script>
```

Supporting Older Browsers

There are actually several versions of JavaScript, and you can specify a version number in your `<script>` tag to avoid confusing older browsers. The following tag specifies JavaScript 1.1:

```
<script language="JavaScript1.1">
```

The current version of JavaScript is 1.5. However, you should not specify a version number at all unless you are using features unique to that version. Throughout this book, we will use `<script>` tags that do not specify a version unless they require a feature of a later version.

Hiding JavaScript Code

Some older browsers, such as Netscape 1.0 and Internet Explorer 1.0, don't support JavaScript at all. There are two important ways to accommodate these browsers when you use JavaScript.

The `<noscript>` Tag

The `<noscript>` tag is used to display a message to non-JavaScript browsers. This includes older browsers as well as current browsers with JavaScript turned off. To use it, simply place text between the opening and closing `<noscript>` tags:

```
<noscript>Your browser doesn't support JavaScript.</noscript>
```

Browsers that do support JavaScript ignore the text between the `<noscript>` tags. Older browsers ignore the `<noscript>` tags and display the text.

Unfortunately, the `<noscript>` tag isn't perfect. Netscape 2.0 doesn't recognize this tag, even though it supports basic JavaScript.

Using HTML Comments

One problem with older browsers is that they might ignore the `<script>` tags and display your JavaScript code in the body of the page. To avoid this, you can enclose your JavaScript code within HTML comments:

```
<script language="JavaScript">
  <!--
  document.write("Your browser supports JavaScript.");
  // -->
</script>
```

The `<!--` and `-->` tags begin and end the HTML comment. This will prevent older browsers from displaying the script, while allowing the script to work on browsers that support it. The `//` in the last line is a JavaScript comment—this prevents the HTML comment tag from being detected as a JavaScript error.

This technique for hiding scripts is not perfect. Certain characters, such as the > (greater-than) symbol, might end the comment prematurely if they appear in your script.

JavaScript Statements

Within the `<script>` tags, you can use one or more JavaScript *statements*, or commands, to create a script. Each statement should end with a semicolon (`;`). Here's a simple example that displays an alert message:

```
<script language="JavaScript">
alert("Hello!");
</script>
```

Most browsers treat the semicolons as optional, but if you include them it will make debugging easier.

Creating Event Handlers

Often, you won't even need to use the `<script>` tag to include JavaScript in a document. Instead, you can use *event handlers*. These are special HTML 4.0 attributes for HTML tags, and can be used to respond to *events*: things that happen in the browser.

For example, the `onMouseOver` event for an HTML tag occurs when the user moves the mouse pointer over that portion of the page. The following HTML defines a link with an `onMouseOver` event handler:

```
<a href="next.html" onMouseOver="alert('hello!');">
```

Within the quotation marks for the event handler, you can use one or more JavaScript statements, separated by semicolons. You will use an event handler like this in the Workshop section of this chapter.

There are actually many events available. Since these are a key part of DHTML, Hour 8, "Responding to Events," discusses all of the events in detail.

Understanding Variables

JavaScript *variables* are virtual containers you can use to store data: a number, a string of text, or something else. For example, the following statement assigns the variable `score` a value of 10:

```
score = 10;
```

JavaScript variable names (and most commands) are case-sensitive. Once you name a variable, be sure you use the same name consistently.

Declaring Variables

JavaScript includes a command, `var`, that you can use to declare variables. This is a way of announcing that a variable will be used and, optionally, assigning it a value. Here's an example:

```
var b=5;
```

This statement creates a variable, `b`, and assigns it an initial value of 5. Most of the time, you can omit the `var` keyword and simply use a variable. Whether you need to use this keyword depends on the variable's *scope*:

- *Local variables* are defined within a function, and can be used only within that function.
- *Global variables* are defined outside any function, and can be used anywhere.

The `var` keyword is only truly required when you declare a local variable within a function that has the same name as a global variable. Using `var` will insure that a local variable is created rather than assigning a new value to the global one.

Functions are groups of commands that you can treat as a unit. They are explained in detail later in this hour.

If you're unsure whether or not to use `var` , you can simply always use it the first time you assign a value to a variable. This makes it easy to avoid errors and will never cause a problem.

Using Arrays

When you need to store a number of similar values, you can use an *array*. Arrays are groups of numbered variables. Each variable within the array has a unique *index*, and is called an *element* of the array.

Unlike variables, arrays must be declared. You declare an array using the `new` keyword. For example, the following statement declares an array called `subtotals` with 10 elements:

```
subtotals = new Array(10);
```

After the array is set up, you can refer to any element just like a variable. For example, this statement assigns the value 5 to the first element of the array:

```
subtotals[0] = 5;
```

Array indices begin with zero. Thus, the ten elements in the `subtotals` array in the example would have the indices `0` through `9`.

Using Strings

So far, you've seen variables used to store numbers. Variables can also store strings of text. For example, the following statement stores the phrase "No time to lose" in a string variable called `phrase`:

```
phrase="No time to lose";
```

Many built-in JavaScript functions use strings. For example, the `document.write` function displays text as part of the current Web page:

```
document.write("This text came from JavaScript.");
```

Using Conditions and Loops

You can accomplish quite a bit using simple JavaScript commands, but for more complex applications you need ways of controlling how your script executes. The following sections will introduce you to conditions and loops, two important ways of controlling your scripts.

Using the `if` Keyword

The `if` statement allows you to test conditions. For example, your script can display an alert if a variable has a certain value:

```
if (score == 0) alert("You lose!");
```

An `if` statement has two parts: a condition (`score==0` in the example) and an action (the `alert` statement in the example.) If the action is a single statement like this, nothing else is required. If you need more than one statement in the action, you enclose them in braces (`{}`):

```
if (score==10) {
   alert("You win!");
   score=0;
} else {
   alert("You lose!");
}
```

This example checks the `score` variable. If it is equal to 10, it displays an alert and resets `score` to zero. It also includes the `else` clause. This clause allows you to specify what should happen if the condition of the `if` statement was *not* true.

Conditional Operators

In the previous example, you might have noticed the double-equals sign (==). This is the *equality operator*, and detects whether two values are equal. JavaScript includes a number of other conditional operators:

- == (is equal to)
- != (is not equal to)
- < (is less than)
- <= (is less than or equal to)
- > (is greater than)
- >= (is greater than or equal to)

A common JavaScript error is to confuse the equality operator (==) with the assignment operator (=). Always use = to assign a value to a variable, and == to test conditions.

Using `for` Loops

One of the nice things about programming languages is that you can let the computer handle repetitive tasks. You can use the `for` keyword in JavaScript to execute a block of statements a number of times. Here is a simple example:

```
for (i=1; i<10; i++) {
  //statements to repeat go here
}
```

The `for` statement includes three elements within the parentheses, separated by semicolons. These include the following:

- An initial value for a variable (`i=1` in the example)
- A condition that must remain true for the loop to continue (`i<10` in the example)
- An *increment expression* to change the value of the variable (`i++` in the example)

The `i++` in the example adds one to the value of `i`. `++` is called the *increment operator*. You can also use the *decrement operator*, `--`, to subtract one from a variable's value.

The `for` statement is followed by a block of statements enclosed in braces. The statements are executed repeatedly as the variable changes, until the condition is no longer met. As an example, the following `for` loop displays the numbers 1 through 10:

```
for (i=1; i<=10; i++) {
   document.write(i + " ");
}
```

Using `while` Loops

Sometimes, rather than executing statements a certain number of times, you'll want them to continue to execute until something happens. You can use the `while` keyword to create this type of loop. Here's an example:

```
while (score < 10) {
  //statements to repeat go here
}
```

This statement would execute the block of code in braces over and over until the variable `score`'s value reached 10 or more. The loop itself doesn't change the variable's value.

In the example above, if none of the statements within the `while` loop change the value of the `score` variable, the loop will never exit. This is called an *infinite loop*, and should be avoided. This type of loop can even crash some browsers.

Using Functions

Functions are groups of JavaScript statements that have been combined under a single name. You've already used built-in functions, such as `document.write`. You can also define your own functions. This allows you to create modular programs, and to repeat the same statements with different values.

Defining a Function

To define a function, you use the `function` keyword. The following statements define a short function:

```
function Linebreak() {
   document.write("<br>");
}
```

This function outputs a single HTML line break (using the `<br>` tag). While this uses a single statement, the true power of functions is to execute several statements at once.

Calling Functions

Once you've defined a function, you can *call* the function from any script on the same page. To call a function, simply use its name followed by parentheses. This statement calls the `Linebreak` function you created earlier:

```
Linebreak();
```

Using Arguments

Functions can have one or more *arguments*, or variables passed to the function. To define a function with arguments, you include one or more variable names within the parentheses. If you are using more than one argument, separate them with commas.

For example, the following is the definition for a function to display an HTML paragraph:

```
function Para(text) {
   document.write("<p>" + text + "</p>");
}
```

This function displays the string you send it in the `text` variable, with paragraph tags on either side. To call this function, simply specify the value of `text` in the parentheses:

```
Para("Welcome to my paragraph.");
```

Variables you use for function arguments are always local variables, and can't be used outside the function.

Returning Values

Functions can also return a value to the script that called the function. This allows you to create functions that answer questions. For example, here is the definition for a function that returns the average of three numbers:

```
function Average(a,b,c) {
  total = a + b + c;
  return total / 3;
}
```

This function accepts three arguments, `a`, `b`, and `c`. It uses the variable `total` to add the numbers, then divides the total by three to get the average. The `return` keyword returns a value. Here is an example of a script to call the `Average` function and display the result:

```
z = Average(2,4,6);
document.write("The average is " + z);
```

To use a function, it must be defined in the same HTML document with which you are working. The best place to define functions is in the `<head>` section of the document, since the definitions themselves don't create any output.

Understanding Objects

JavaScript also includes *objects*, a special type of variable that can store multiple data items and even functions to work with those items. In fact, strings and arrays in JavaScript are just special types of objects.

Object Properties and Methods

JavaScript supports a number of built-in objects, including those in the DOM. Objects can have *properties*, or variables associated with the object. Object and property names are separated by periods.

For example, the `location.href` property contains the current document's URL. You can change this property to force the browser to go to another Web page:

```
location.href="http://www.mcp.com/";
```

Objects can also have *methods*, or functions associated with the object. You've already used one of these: The `document.write` function is actually the `write` method of the `document` object.

Appendix D includes a summary of the objects, such as document, in the Level 0 DOM. You'll begin to learn about the objects in the W3C DOM, their properties and methods in Hour 5, "Understanding the DOM."

Workshop: Adding a Script to a Web Page

You should now have a basic understanding of JavaScript. To bring together what you've learned, you will now create a simple script and get it working on a Web page.

In the previous hour's Workshop section, you created a simple HTML document for the Figby Industries Web page. You can now use JavaScript to add a feature: descriptions that appear in the status line when the user moves the mouse pointer over the links in the page.

This hour has presented a quick introduction to JavaScript, but there's much more to learn. See Appendix A for a list of Web sites and books with further information.

Adding Event Handlers

To add descriptions to the links, you can use two event handlers:

- `onMouseOver`—occurs when the mouse pointer moves over an object
- `onMouseOut`—occurs when the mouse pointer leaves an object

The listing below shows one of the links from the Figby Industries page with both of these event handlers added.

```
<a href="products.html"
  onMouseOver="window.status='Learn about our products';return true;"
  onMouseOut="window.status='';return true;">
  <b>Products</b></a>
```

The `onMouseOver` event handler assigns a value to the `window.status` property, which displays the message in the status line. The `return true` statement tells the browser to keep this message in the status line rather than rewriting it with the link's URL.

The `window.status` property is part of the Level 0 DOM. See Appendix D for a summary of Level 0 DOM objects, properties and methods.

The `onMouseOut` event handler assigns a null string to `window.status` to clear the status line.

Putting It All Together

By adding event handlers to the links in the original Figby Industries page, you can make the status line display helpful descriptions. Listing 3.1 shows the complete example with all of the event handlers.

LISTING 3.1 The Complete JavaScript Example

```
<html>
<head>
   <title>Figby Industries, Inc.</title>
</head>
```

LISTING 3.1 continued

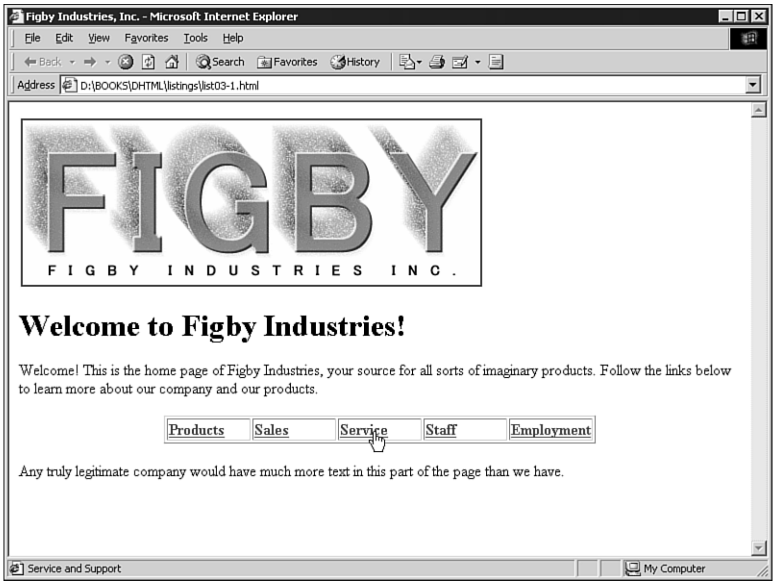
```
<body>
<img align="center" src="logo.gif"
width="486" height="180" border="0" alt=""><br>
<h1>Welcome to Figby Industries!</h1>
<p>Welcome! This is the home page of Figby Industries,
your source for all sorts of imaginary products. Follow the links
below to learn more about our company and our products.
</p>
<table border="1" align="center">
<tr>
  <td width="20%"><a href="products.html"
   onMouseOver="window.status='Learn about our products';return true;"
   onMouseOut="window.status='';return true;">
   <b>Products</b></a></td>
  <td width="20%"><a href="sales.html"
   onMouseOver="window.status='Contact our sales department';return true;"
   onMouseOut="window.status='';return true;">
   <b>Sales</b></a></td>
  <td width="20%"><a href="service.html"
   onMouseOver="window.status='Service and Support';return true;"
   onMouseOut="window.status='';return true;">
   <b>Service</b></a></td>
  <td width="20%"><a href="staff.html"
   onMouseOver="window.status='Meet our staff';return true;"
   onMouseOut="window.status='';return true;">
   <b>Staff</b></a></td>
  <td width="20%"><a href="jobs.html"
   onMouseOver="window.status='Employment opportunities';return true;"
   onMouseOut="window.status='';return true;">
   <b>Employment</b></a></td>
</tr>
</table>
<p>Any truly legitimate company would have much more text in this
part of the page than we have.</p>
</body>
</html>
```

In this document, the links within the table include `onMouseOver` and `onMouseOut` event handlers. Since all of the functions are performed within event handlers, there's no need for the `<script>` tag at all.

This example requires an image, `logo.gif`. As with the other examples in this book, you can download this image and the HTML document itself from this book's Web site, `http://www.starlingtech.com/dhtml/`.

To try this example, save it as an HTML document and load it into a browser. Figure 3.1 shows Internet Explorer's display of this example. In the figure, the mouse pointer is over the "Service" link.

FIGURE 3.1
Internet Explorer shows the complete JavaScript example.


Figby Industries, Inc. - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address D:\BOOKS\DHTML\listings\list03-1.html
FIGBY
FIGBY INDUSTRIES INC.
Service and Support
My Computer

Welcome to Figby Industries!

Welcome! This is the home page of Figby Industries, your source for all sorts of imaginary products. Follow the links below to learn more about our company and our products.

Products	Sales	Service	Staff	Employment

Any truly legitimate company would have much more text in this part of the page than we have.

Summary

In this hour, you've reviewed the basics of JavaScript: its syntax, and how to use variables, loops, functions, and conditional statements. You've also completed a simple JavaScript example.

In the next hour, you'll get down to business by creating a DHTML example.

Q&A

Q. Can JavaScript display text on a Web page after it's already loaded?

A. No, but Dynamic HTML can do this and much more. You'll use DHTML to change text within a page in the next hour.

Q. I already know Microsoft's VBScript language. Can I use it with DHTML?

A. Yes, VBScript includes access to the DOM. However, your applications will only work in Internet Explorer.

Q. What's the difference between JavaScript and JScript?

A. JScript is Microsoft's implementation of JavaScript. For most purposes, it is compatible with JavaScript. All of the JavaScript examples in this book will work in Internet Explorer or Netscape.

Q. Can I hide my JavaScript code from users using the View Source option?

A. No. While there are a few complicated methods that obscure your code, a determined user can always view the source of your HTML document and its associated JavaScript code.

Quiz

Test your knowledge of the material covered in this hour by answering the following questions.

Questions

1. Which JavaScript keyword would be the best choice if you need to create a loop that executes exactly five times?
   a. `while`
   b. `for`
   c. `wherefore`
2. Which of the following `if` statements correctly checks the variable `a` for the value of `20`?
   a. `if a==20`
   b. `if (a=20)`
   c. `if (a==20)`
3. Which type of JavaScript variable is ideal for storing a series of numbers?
   a. array
   b. string
   c. numeric variable

Answers

1. b. A `for` loop would be the best way to execute a loop exactly five times.
2. c. The correct statement is `if (a==20)`. Notice that the equality operator (`==`) must be used, and the parentheses around the condition are required.
3. a. A JavaScript array is ideal for storing a series of numbers.

Exercises

If you'd like to practice using JavaScript before you continue, try the following exercises:

- Modify Listing 3.1 to use a function to handle the status line instead of assigning `window.status` directly. You can use the text for the status line as an argument to the function.
- Try including the `average` function described earlier in this hour to the `<head>` section of an HTML document. Add a `<script>` section within the body of the document to display the average of the numbers 7, 12, and 20.