1

WELCOME TO JAVASCRIPT!

Welcome to *The Book of JavaScript*. JavaScript is one of the fastest and easiest ways to make your Web site truly dynamic. If you want to spruce up tired-looking pages, you've got the right book. This book will give you ready-made JavaScripts you can implement immediately on your Web site, and I'll take you step by step through sample scripts (both hypothetical and real-world examples) so you *understand* how JavaScript works. With this understanding, you can modify scripts to fit your specific needs as well as write scripts from scratch. Your knowledge of JavaScript will grow as you work through the book; each chapter introduces and explores in depth a new JavaScript topic by highlighting a real-life Web site that uses it.

Is JavaScript for You?

If you want a quick, easy way to add interactivity to your Web site, if the thought of using complex programs intimidates you, or if you're simply interested in programming but don't know where to start, JavaScript is for you.

JavaScript, a programming language built into your Web browser, is one of the best ways to add interactivity to your Web site because it's the only cross-browser language that works directly with Web browsers. Other languages, such as Java, Perl, and C, don't have direct access to the images, forms, and windows that make up a Web page.

JavaScript is also very easy to learn. You don't need any special hardware or software, you don't need access to a Web server, and you don't need a degree in computer science to get things working. All you need is a Web browser and a text editor such as SimpleText or Notepad.

Finally, JavaScript is a *complete* programming language, so if you want to learn more about programming, it provides a great introduction. If you don't give a hoot about programming, that's fine too. There are plenty of places—including this book and its bundled CD-ROM—where you can get prefab scripts to cut and paste right into your pages.

Is This Book for You?

This book assumes you don't have any programming background. Even if you have programmed before, you'll find enough that's new in JavaScript to keep you entertained. One of the best things about JavaScript is that you don't have to be mega-geeky to get it working on your Web pages right away. You *do* need a working knowledge of HTML, however.

The Goals of This Book

The main goal of this book is to get you to the point of writing your own JavaScripts. Equally important, however, is the ability to read other people's scripts. JavaScript is a sprawling language, and you can learn thousands of little tricks from other scripts. In fact, once you've finished this book, you'll find viewing the source code of JavaScript Web pages the best way to increase your knowledge.

Each of the following chapters includes JavaScript examples from professional sites. You'll soon see there are many ways to script, and sometimes going through the site's code unearths interesting corners of JavaScript that I don't cover in this book.

Beyond learning how to write your own JavaScript and read other people's scripts, I also want you to learn where to look for additional information on JavaScript. The best place to learn new techniques is to view the source of a Web page you find interesting. Several Web sites also offer free JavaScripts. I'll be introducing some of these as we go along, but here are a few good examples to get you started:

http://www.js-planet.com

http://www.developer.earthweb.com/directories/pages/dir.javascript.html

http://www.javascript.com

Another good place to get information is a JavaScript reference book. *The Book of JavaScript* is primarily a tutorial for learning basic JavaScript and making your Web site interactive. It's not a complete guide to the language, which includes too many details for even a 400-page introduction to cover. If you're planning to become a true JavaScript master, I suggest picking up *JavaScript: The Definitive Guide* by David Flanagan (O'Reilly and Associates, 1998) after making your way through this book. The last 500 or so pages of Flanagan's book not only list every JavaScript command, but also tell you in which browsers they work.

What Can JavaScript Do?

JavaScript can add interactivity to your Web pages in a number of ways. This book offers many examples of JavaScript's broad capabilities. The following are just two examples that illustrate what you can do with JavaScript.

The first example (Figure 1-1) is a flashing grid of colored squares (to get the full effect, visit http://hotwired.lycos.com/webmonkey/demo/96/35/index4a.html or the bundled CD-ROM under chapter01/fig1-1.html) created by a fellow named Taylor. Flashy, isn't it? In this example, a JavaScript function changes the color of a randomly chosen square in the grid every second or so.

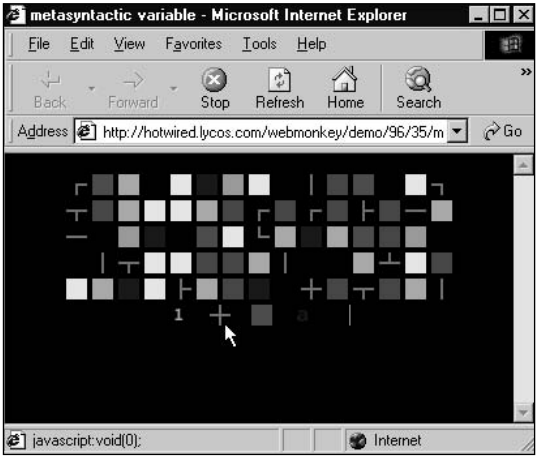

Figure 1-1: A demonstration of JavaScript's artful abilities

Mousing over one of the five icons below the squares (number, plus sign, square, letter, and horizontal line) tells the page to use that type of image for the new images appearing on the grid. For example, mousing over the number icon tells the JavaScript to start replacing the squares with 1s and 0s instead of different-colored squares. This page illustrates four important JavaScript features you'll learn about throughout the book: how to change images on a Web page, affect Web pages over time, add randomness to Web pages, and dynamically change what's happening on a Web page based on an action someone viewing the page takes.

Although Taylor's demo is beautiful, it's not the most useful application of JavaScript. Figure 1-2 (available on the CD-ROM in chapter1/figure1-2.html) shows you a *much* more useful page that calculates the weight of a fish based on its length. Enter the length and type of fish, and the JavaScript calculates the fish's weight. This fishy code demonstrates JavaScript's ability to read what a visitor has entered into a form, perform a mathematical calculation based on the input, and provide feedback by displaying the results in another part of the form. Maybe calculating a fish's weight isn't the most useful application of JavaScript either, but you can use the same skills to calculate a monthly payment on a loan (Chapter 7), score a quiz (Chapter 10), or verify that a visitor has provided a valid email address (Chapter 11).

These are just two examples of the many features JavaScript can add to your Web sites. Each chapter will cover at least one new application. If you want a preview of what you'll learn, read the first page or so of each chapter.

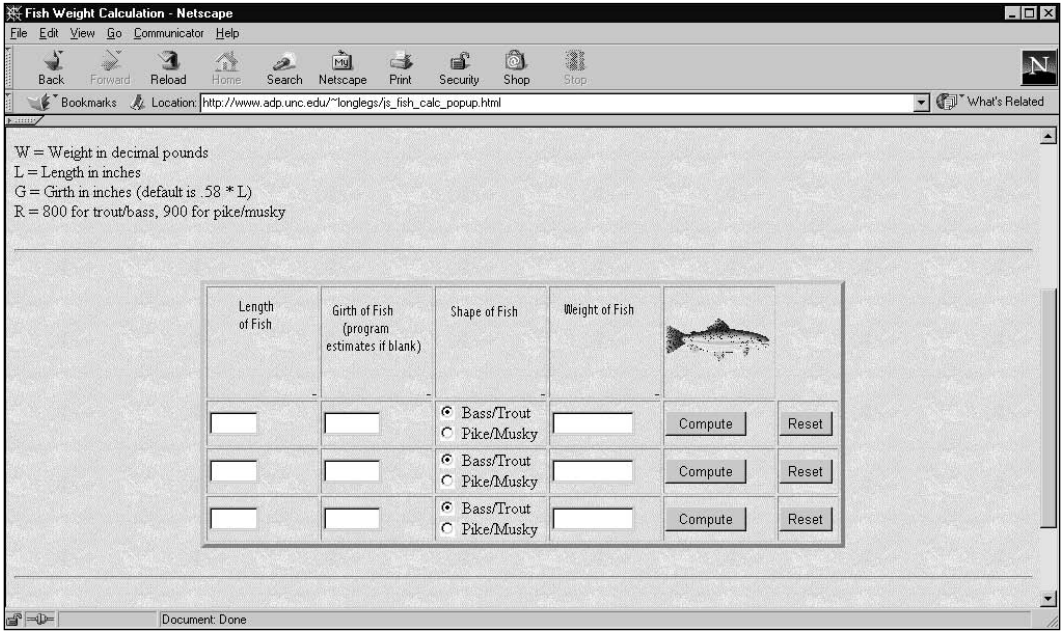

Figure 1-2: How much does my fish weigh?

What Are the Alternatives to JavaScript?

Several other programming languages can add interactivity to Web pages, but they all differ from JavaScript in important ways. The three main alternatives are CGI scripting, Java, and VBScript.

CGI Scripting

Before JavaScript, using CGI scripts was the only way to make Web pages do more than hyperlink to other Web pages. *CGI* (Common Gateway Interface—a language browsers and programs can use to communicate) scripts allow Web pages to send messages to computer programs on Web servers. For example, a Web browser can send information (perhaps the length and type of fish) to a CGI program that calculates the weight of fish. The program runs its calculations, formats the answer as an HTML page, and sends the answer back to the browser.

CGI scripts are very powerful, but because they reside on the Web server, they have some drawbacks. First, the connection between your Web browser and the Web server limits the speed of your Web page's interactivity. This may not sound like a big problem, but imagine the following scenario: You're filling out an order form with a dozen entry fields (such as name, address, and phone number—see Figure 1-3 on page 6), but you forget to include your phone number and address. When you press the Submit button to send the information across the Internet to the Web server, the Web server sees that you didn't fill out the form completely and sends a message back across the Internet requesting that you finish the job. This cycle could take quite a while over a slow connection. If you fill out the form incorrectly again, you have to wait through another cycle. People find this process tiresome, especially if they're customers who want their orders processed quickly.

With JavaScript, though, the programs you write run in the browser itself. This means the browser can make sure you've filled out the form correctly *before* sending the form's contents to the Web server. JavaScript reduces the time your information spends traveling between the browser and the server by waiting until you're really done before sending information.

Another drawback to CGI scripts is that the Web server running a CGI program can get bogged down if too many people use the program simultaneously. Serving up HTML pages is pretty easy for a Web server. However, some CGI programs take a long time to run on a machine, and each time someone tries to run the program, the server has to start up another copy of it. As more and more people try to run the program, the server slows down progressively. If 1,000 people run the program, the server might take so long to respond that some browsers give up, think the server is dead, and give visitors a "server busy" error. This problem doesn't exist in JavaScript because its programs run on each visitor's Web browser—not on the Web server.

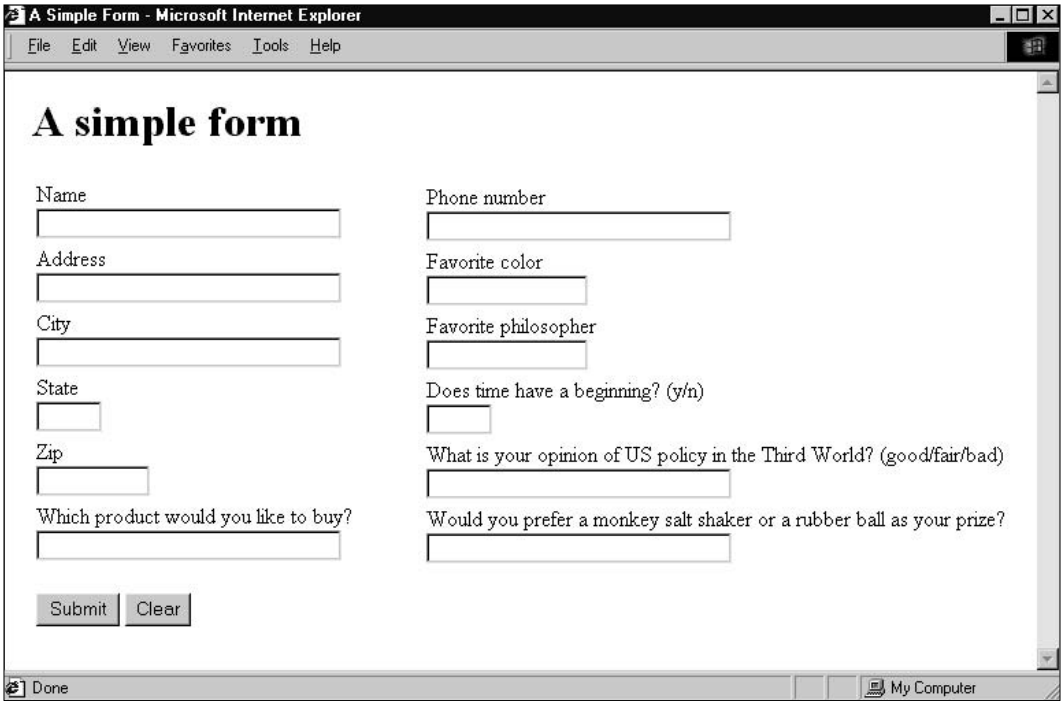

Figure 1-3: A simple order form

A third problem with CGI scripts is that not everyone has access to the parts of a Web server that can run CGI scripts. Since a CGI script can conceivably crash a Web server, engineers generally guard this area, only allowing fellow engineers access. JavaScript, on the other hand, goes right into the HTML of a Web page. If you can write a Web page, you can put JavaScript in the page without permission from recalcitrant engineers.

Java

Although JavaScript and Java have similar names, they aren't the same. Netscape initially created JavaScript to provide interactivity for Web pages, and Sun Microsystems wrote Java as a general programming language that works on all kinds of operating systems. As you'll see in Appendix A, though JavaScript can talk to Java programs, they're two totally separate languages.

VBScript

The language most similar to JavaScript is Microsoft's proprietary language, VBScript (VB stands for Visual Basic). Like JavaScript, VBScript runs on your Web browser and adds interactivity to Web pages. However, VBScript works only on computers running Microsoft Internet Explorer on Microsoft Win-

dows, so unless you want to restrict your readership to people who use *that* browser on *that* operating system, you should go with JavaScript.

JavaScript's Limitations

Yes, JavaScript does have limitations—but they derive from its main purpose: to add interactivity to your Web pages.

JavaScript Can't Talk to Servers

One of JavaScript's drawbacks is also its main strength: It works entirely within the Web browser. As we've seen, this cuts down on the amount of time your browser spends communicating with a server. On the other hand, JavaScript can't handle some server tasks you may need to do.

The two primary tasks JavaScript can't handle are aggregating information from your users and communicating with other machines. If you want to write a survey that asks your visitors a couple of questions, stores their answers in a database, and sends an email thanks when they finish, you'll have to use a program on your server. As we'll see in Chapter 7, JavaScript *can* make the survey run more smoothly, but once a visitor has finished it, JavaScript can't store the information on the server—JavaScript works only on the browser and doesn't contact the server. In order to store the survey information, you need a CGI script or a Java applet, both of which fall outside the scope of this book.

Sending email with JavaScript is also a problem: Your JavaScript would have to contact a server to send the mail, and of course it can't do this. Again, you need a CGI script or Java applet for this job.

JavaScript Can't Create Graphics

Another of JavaScript's limitations is that it can't create its own graphics. Whereas more complicated languages can draw pictures, JavaScript can only manipulate existing pictures (that is, GIF or JPEG files). Luckily, because JavaScript can manipulate created images in so many ways, you shouldn't find this too limiting.

JavaScript Works Differently in Different Browsers

Perhaps the most annoying problem with JavaScript is that it works somewhat differently in various browsers. JavaScript first came out with Netscape 2.0 and has changed slightly with each new version of Netscape. Microsoft started including its own variation with Microsoft Internet Explorer 3.0 (MSIE) and added some features to its version of JavaScript when it introduced MSIE 4.0. Luckily, the core of JavaScript is identical in all the browsers; mostly the fancy stuff like Dynamic HTML (Chapter 13) is different. I've taken Netscape 3.0's JavaScript as the standard and I'll point out incompatibilities with other browsers as they come up throughout the book.

Getting Started

We're about ready to begin. You need a Web browser and a text editor. Any text editor will do: Notepad or Wordpad in Windows and SimpleText on a Macintosh are the simplest choices. Microsoft Word or Corel's WordPerfect will work as well. You can also use a text editor such as BBEdit and HomeSite—these are designed to work with HTML and JavaScript.

NOTE *Always save documents as text only and end their names with .html or .htm. Unless you tell Microsoft Word and WordPerfect otherwise, both programs write documents in formats Web browsers can't read. If you try to open a Web page you've written and the browser shows a lot of weird characters you didn't put in your document, go back and make sure you've saved it as text only.*

Some tools for building Web sites will actually write JavaScript for you—for example, Macromedia's DreamWeaver and GoLive's Cyberstudio 5.0. These tools work fine when you want to write common JavaScripts such as image rollovers and you know you'll never want to change them. Unfortunately, the JavaScript often ends up much longer than necessary, and you may find it difficult to understand and change to suit your needs. Unless you want a JavaScript that works *exactly* like one provided by the package you've purchased, you're often best off writing scripts by hand. However, these tools do a great job when you want to whip up a prototype. Once you've used the tool to figure out how you want your page to behave, you can go back and rewrite the script.

Where JavaScript Goes on Your Web Pages

Now let's get down to some JavaScript basics. Figure 1-4 shows you the thinnest possible skeleton of an HTML page with JavaScript. The numbered lines are JavaScript.

In Figure 1-4, you can see the JavaScript between the `<script language="JavaScript">` (`1`) and `</script>` (`4`) tags. With one exception, which Chapter 4 will cover, all JavaScript goes between these tags.

NOTE *You can't include any HTML between the open* `<script>` *and close* `</script>` *tags. While you're inside those tags, your browser assumes everything it sees is JavaScript. If it sees HTML in there, it gets confused and gives you an error message.*

The JavaScript tags can go in either the head or the body of your HTML page. It doesn't matter too much where you put them, although you're generally best off putting as much JavaScript in the head as possible. That way you don't have to look for it all over your Web pages.

```
   <html>
   <head>
   <title>JavaScript Skeleton</title>
1  <script language="JavaScript">
2  // JavaScript can go here!
3  // But no HTML!
4  </script>
   </head>
   <body>
5  <script language="JavaScript">
6  // JavaScript can go here too!
7  // But no HTML!
8  </script>
   </body>
   </html>
```

Figure 1-4: An HTML page with JavaScript

The lines that start with two slashes (2, 3, 6, and 7) are JavaScript comments. The browser ignores any text that appears after two slashes. Comments are extremely important because programming languages just aren't natural. The script you're writing may make perfect sense while you're writing it, but a few days later when you want to make a little modification, you might spend hours just figuring out what you wrote the first time. If you comment your code, you'll save yourself the hassle of trying to remember what you were thinking when you wrote that bizarre code at 2 a.m. in the midst of what seemed like an amazingly lucid caffeine haze.

Dealing with Older Browsers

There's a slight problem with the JavaScript skeleton in Figure 1-4 (other than the fact that it doesn't really have any JavaScript in it): Netscape didn't introduce the `<script>` tag until version 2.0, so any Netscape browser older than that won't recognize the tag.

When a browser sees an HTML tag it doesn't understand, it just ignores that tag. That's generally a good thing. However, a browser that doesn't understand JavaScript will write your lines of JavaScript to the browser as text. Figure 1-5 on page 10 shows what the JavaScript skeleton in Figure 1-4 would look like in an older browser.

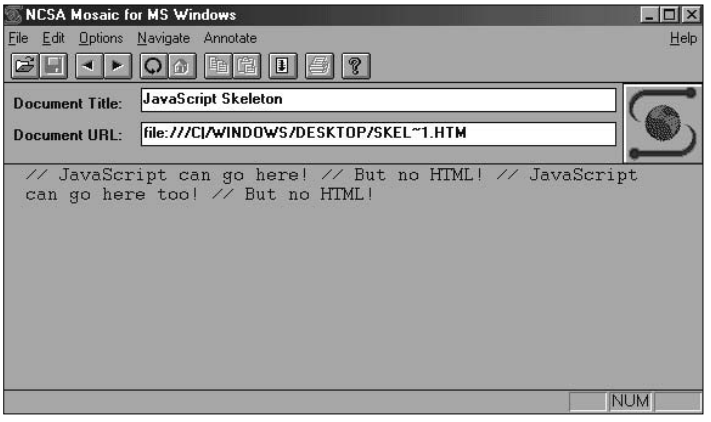

Figure 1-5: What Figure 1-4 would look like in an older browser

Hiding JavaScript from Older Browsers

For those who still use browsers like Microsoft Internet Explorer 2.0 or MacWeb 0.9, it's nice to add the code in 1 and 2 of Figure 1-6, which hides your JavaScript from them.

```
   <script language="JavaScript">
1  <!-- hide me from older browsers

   // JavaScript goes here

2  // end hiding stuff -->
   </script>
```

Figure 1-6: Hiding JavaScript from browsers that don't understand it

The important symbols are the `<!--` code at 1 and the `// -->` comments at 2. These weird lines work because older browsers see the `<!--` and `-->` code as marking an entire *block* of HTML comments, and just ignore everything between them. Luckily, browsers that understand JavaScript have different rules for HTML comments: They see everything from `<!--` to the end of that *line* as a comment, not the whole block between 1 and 2, so they won't ignore the JavaScript. The words in the tags ("hide me from older browsers" and "end hiding stuff") aren't important. You can make those whatever you want, or just leave them out entirely. It's the `<!--` and `// -->` tags that are important.

Frankly, this is a bit tough to understand. If you're having trouble wrapping your mind around why it works, don't worry—just remember to put the `<!--` tag on its own line right after `<script>` and the `// -->` tag on its own line right before `</script>`, and people with older browsers will thank you.

Your First JavaScript

It's time to run your first JavaScript program. I'll explain the code in Figure 1-7 in the next chapter, so for now just type the code into your text editor, save it as **`my_first_program.html`**, and then run it in your browser. If you don't want to type it all in, run the example from the CD-ROM under chapter01/fig1-7.html.

```
<html>
<head>
<title>JavaScript Skeleton</title>
</head>
<body>
<script language="JavaScript">
<!-- hide this from older browsers
// say hello world!
1 alert("hello world!");
// end hiding comment -->
</script>
</body>
</html>
```

Figure 1-7: Your first JavaScript program

When a browser reads this Web page, the JavaScript at `1` instructs the browser to put up a little window with the words `hello world!` in it. Figure 1-8 shows you what this looks like. Traditionally, this is the first script you write in any programming language. It gets you warmed up for the fun to come.

Figure 1-8: The "hello world!" script

Summary

Congratulations—you're now on your way to becoming a bona fide JavaScripter! This chapter has given you all the basic tools you need and has shown you how to get a very basic JavaScript program running. If you followed everything here, you now know:

- Some of the great things JavaScript can do.
- How JavaScript compares to CGI scripting, VBScript, and Java.
- JavaScript's main limitations.
- Where JavaScript goes on the page.
- How to write JavaScript older browsers won't misunderstand.

If you understood all that, you're ready to write some code!

Assignment

If you haven't tried typing Figure 1-7 into a text editor and running it in a Web browser, try it. You'll find next chapter's assignments hard to do if you can't get Figure 1-7 to work. If you're sure you've recreated Figure 1-7 exactly and it's not working, make sure you're saving the file as text-only. You may also find it helpful to peruse Chapter 14, which discusses ways to fix broken code. Although you may not understand everything in that chapter, you may find some helpful tips. It may also be wise to run the version of Figure 1-7 that is on your CD-ROM under chapter01/fig1-7.html. If that doesn't work, you may be using a browser that doesn't support JavaScript, or your browser may be set to reject JavaScript. If you're sure you're using a browser that supports JavaScript (Netscape 2.0 and higher versions, and Microsoft Internet Explorer 3.0 and higher), check your browser's options and make sure it's set to run JavaScript.