

1

Introduction to JavaScript and the Web

In this introductory chapter, you look at what JavaScript is, what it can do for you, and what you need in order to use it. With these foundations in place, you will see throughout the rest of the book how JavaScript can help you to create powerful web applications for your web site.

The easiest way to learn something is by actually doing it, so throughout the book you'll create a number of useful example programs using JavaScript. This process starts in this chapter, by the end of which you will have created your first piece of JavaScript code.

Additionally, over the course of the book you'll develop a complete JavaScript web application: an online trivia quiz. By seeing it develop, step by step, you'll get a good understanding of how to create your own web applications. At the end of this chapter, you'll look at the finished trivia quiz and consider the ideas behind its design.

Introduction to JavaScript

In this section you take a brief look at what JavaScript is, where it came from, how it works, and what sorts of useful things you can do with it.

What Is JavaScript?

Having bought this book, you are probably already well aware that JavaScript is some sort of *computer language*, but what is a computer language? Put simply, a computer language is a series of instructions that tell the computer to do something. That something can be one of a wide variety of things, including displaying text, moving an image, or asking the user for information. Normally the instructions, or what is termed *code*, are *processed* from the top line downward. This simply means that the computer looks at the code you've written, works out what action you want taken, and then takes that action. The actual act of processing the code is called *running* or *executing* it.

Chapter 1: Introduction to JavaScript and the Web

In natural English, here are instructions, or code, you might write to make a cup of instant coffee:

1. Put coffee crystals in cup.
2. Fill kettle with water.
3. Put kettle on to boil.
4. Has the kettle boiled? If so, then pour water into cup; otherwise, continue to wait.
5. Drink coffee.

You'd start running this code from the first line (instruction 1), and then continue to the next (instruction 2), then the next, and so on until you came to the end. This is pretty much how most computer languages work, JavaScript included. However, there are occasions when you might change the flow of execution, or even skip over some code, but you'll see more of this in Chapter 3.

JavaScript is an interpreted language, rather than a compiled language. What is meant by the terms *interpreted* and *compiled*?

Well, to let you in on a secret, your computer doesn't really understand JavaScript at all. It needs something to interpret the JavaScript code and convert it into something that it understands; hence it is an *interpreted language*. Computers understand only *machine code*, which is essentially a string of binary numbers (that is, a string of zeros and ones). As the browser goes through the JavaScript, it passes it to a special program called an *interpreter*, which converts the JavaScript to the machine code your computer understands. It's a bit like having a translator to translate English into Spanish, for example. The important point to note is that the conversion of the JavaScript happens at the time the code is run; it has to be repeated every time this happens. JavaScript is not the only interpreted language; there are others, including VBScript.

The alternative *compiled language* is one in which the program code is converted to machine code before it's actually run, and this conversion only has to be done once. The programmer uses a compiler to convert the code that he wrote to machine code, and it is this machine code that is run by the program's user. Compiled languages include Visual Basic and C++. Using a real-world analogy, it's like having someone translate your English document into Spanish. Unless you change the document, you can use it without retranslation as much as you like.

Perhaps this is a good point to dispel a widespread myth: JavaScript is not the script version of the Java language. In fact, although they share the same name, that's virtually all they do share. Particularly good news is that JavaScript is much, much easier to learn and use than Java. In fact, languages like JavaScript are the easiest of all languages to learn, but they are still surprisingly powerful.

JavaScript and the Web

For most of this book you'll look at JavaScript code that runs inside a web page loaded into a browser. All you need in order to create these web pages is a text editor — for example, Windows Notepad — and a web browser, such as Firefox or Internet Explorer, with which you can view your pages. These browsers come equipped with JavaScript interpreters.

In fact, the JavaScript language first became available in the web browser Netscape Navigator 2. Initially, it was called LiveScript. However, because Java was the hot technology of the time, Netscape decided

that JavaScript sounded more exciting. When JavaScript really took off, Microsoft decided to add its own brand of JavaScript, called JScript, to Internet Explorer. Since then, Netscape, Microsoft, and others have released improved versions and included them in their latest browsers. Although these different brands and versions of JavaScript have much in common, there are enough differences to cause problems if you're not careful. Initially you'll be creating code that'll work with most browsers, whether Firefox, Internet Explorer, or Netscape. Later chapters look at features available only to Firefox 1.5 or later and Internet Explorer 6 and 7. You'll look into the problems with different browsers and versions of JavaScript later in this chapter, and see how to deal with them.

The majority of the web pages containing JavaScript that you create in this book can be stored on your hard drive and loaded directly into your browser from the hard drive itself, just as you'd load any normal file (such as a text file). However, this is not how web pages are loaded when you browse web sites on the Internet. The Internet is really just one great big network connecting computers. Access to web sites is a special service provided by particular computers on the Internet; the computers providing this service are known as *web servers*.

Basically the job of a web server is to hold lots of web pages on its hard drive. When a browser, usually on a different computer, requests a web page contained on that web server, the web server loads it from its own hard drive and then passes the page back to the requesting computer via a special communications protocol called *Hypertext Transfer Protocol (HTTP)*. The computer running the web browser that makes the request is known as the *client*. Think of the client/server relationship as a bit like a customer/shopkeeper relationship. The customer goes into a shop and says, "Give me one of those." The shopkeeper serves the customer by reaching for the item requested and passing it back to the customer. In a web situation, the client machine running the web browser is like the customer, and the web server providing the page requested is like the shopkeeper.

When you type an address into the web browser, how does it know which web server to get the page from? Well, just as shops have addresses, say, 45 Central Avenue, Sometownsville, so do web servers. Web servers don't have street names; instead they have *Internet protocol (IP) addresses*, which uniquely identify them on the Internet. These consist of four sets of numbers, separated by dots; for example, 127.0.0.1.

If you've ever surfed the net, you're probably wondering what on earth I'm talking about. Surely web servers have nice `www.somewebsite.com` names, not IP addresses? In fact, the `www.somewebsite.com` name is the "friendly" name for the actual IP address; it's a whole lot easier for us humans to remember. On the Internet, the friendly name is converted to the actual IP address by computers called *domain name servers*, which your Internet service provider will have set up for you.

Toward the end of the book, you'll go through the process of setting up your own web server in a step-by-step guide. You'll see that web servers are not just dumb machines that pass pages back to clients, but in fact can do a bit of processing themselves using JavaScript. You'll look at this later in the book as well.

One last thing: Throughout this book we'll be referring to the Internet Explorer browser as IE.

Why Choose JavaScript?

JavaScript is not the only scripting language; there are others such as VBScript and Perl. So why choose JavaScript over the others?

Chapter 1: Introduction to JavaScript and the Web

The main reason for choosing JavaScript is its widespread use and availability. Both of the most commonly used browsers, IE and Firefox, support JavaScript, as do almost all of the less commonly used browsers. So you can assume that most people browsing your web site will have a version of JavaScript installed, though it is possible to use a browser's options to disable it.

Of the other scripting languages we mentioned, VBScript, which can be used for the same purposes as JavaScript, is supported only by Internet Explorer running on the Windows operating system, and Perl is not used at all in web browsers.

JavaScript is also very versatile and not just limited to use within a web page. For example, it can be used in Windows to automate computer-administration tasks and inside Adobe Acrobat .pdf files to control the display of the page just as in web pages, although Acrobat uses a more limited version of JavaScript. However, the question of which scripting language is the most powerful and useful has no real answer. Pretty much everything that can be done in JavaScript can be done in VBScript, and vice versa.

What Can JavaScript Do for Me?

The most common uses of JavaScript are interacting with users, getting information from them, and validating their actions. For example, say you want to put a drop-down menu on the page so that users can choose where they want to go to on your web site. The drop-down menu might be plain old HTML, but it needs JavaScript behind it to actually do something with the user's input. Other examples of using JavaScript for interactions are given by forms, which are used for getting information from the user. Again, these may be plain HTML, but you might want to check the validity of the information that the user is entering. For example, if you had a form taking a user's credit card details in preparation for the online purchase of goods, you'd want to make sure he had actually filled in those details before you sent the goods. You might also want to check that the data being entered are of the correct type, such as a number for his age rather than text.

JavaScript can also be used for various tricks. One example is switching an image in a page for a different one when the user rolls her mouse over it, something often seen in web page menus. Also, if you've ever seen scrolling messages in the browser's status bar (usually at the bottom of the browser window) or inside the page itself and wondered how that works, this is another JavaScript trick that you'll learn about later in the book. You'll also see how to create expanding menus that display a list of choices when a user rolls his or her mouse over them, another commonly seen JavaScript-driven trick.

Tricks are okay up to a point, but even more useful are small applications that provide a real service. For example, a mortgage seller's web site with a JavaScript-driven mortgage calculator, or a web site about financial planning that includes a calculator that works out your tax bill for you. With a little inventiveness, you'll be amazed at what can be achieved.

Tools Needed to Create JavaScript Web Applications

All that you need to get started with creating JavaScript code for web applications is a simple text editor, such as Windows Notepad, or one of the many slightly more advanced text editors that provide line numbering, search and replace, and so on. An alternative is a proper HTML editor; you'll need one that enables you to edit the HTML source code, because that's where you need to add your JavaScript. A

number of very good tools specifically aimed at developing web-based applications, such as the excellent Dreamweaver 8 from Adobe, are also available. However, this book concentrates on JavaScript, rather than any specific development tool. When it comes to learning the basics, it's often best to write the code by hand rather than relying on a tool to do it for you. This helps you to understand the fundamentals of the language before you attempt the more advanced logic that is beyond a tool's capability. When you have a good understanding of the basics, you can use tools as timesavers so that you can spend more time on the more advanced and more interesting coding.

You'll also need a browser to view your web pages in. It's best to develop your JavaScript code on the sort of browsers you expect visitors to use to access your web site. You'll see later in the chapter that there are different versions of JavaScript, each supported by different versions of the web browsers. Each of these JavaScript versions, while having a common core, also contains various extensions to the language. All the examples that we give in this book have been tested on Firefox version 1.5, and IE versions 6 and 7. Wherever a piece of code does not work on any of these browsers, a note to this effect has been made in the text.

If you're running Windows, you'll almost certainly have IE installed. If not, a trip to www.microsoft.com/windows/ie/default.aspx will get you the latest version.

Firefox can be found at www.mozilla.com/firefox/all.html. When installing Firefox it's worth going for the custom setup. This will give you the option later on of choosing which bits to install. In particular it's worth selecting the Developer Tools component. While not essential, it's an extra that's nice to have.

Even if your browser supports JavaScript, it is possible to disable this functionality in the browser. So before you start on your first JavaScript examples in the next section, you should check to make sure JavaScript is enabled in your browser.

To do this in Firefox, choose Options from the Tools menu on the browser. In the window that appears, click the Content tab. From this tab make sure the Enable JavaScript check box is selected, as shown in Figure 1-1.

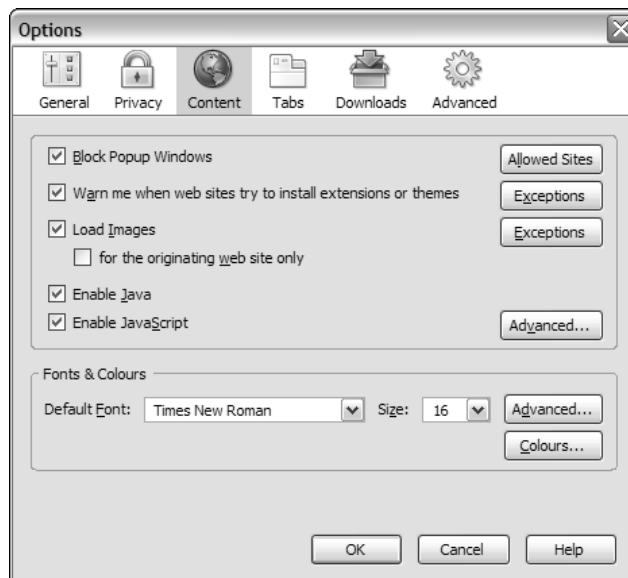


Figure 1-1

Chapter 1: Introduction to JavaScript and the Web

It is harder to turn off scripting in Internet Explorer. Choose Internet Options from the Tools menu on the browser, click the Security tab, and check whether the Internet or Local intranet options have custom security settings. If either of them does, click the Custom Level button and scroll down to the Scripting section. Check that Active Scripting is set to Enable.

A final point to note is how to open the code examples in your browser. For this book, you simply need to open the file on your hard drive in which an example is stored. You can do this in a number of ways. One way in IE6 is to choose Open from the File menu, and click the Browse button to browse to where you stored the code. Similarly, in Firefox choose Open File from the File menu, browse to the file you want, and click the Choose File button.

IE7, however, has a new menu structure and this doesn't include an Open File option. You can get around this by typing the drive letter of your hard drive followed by a colon in the address bar, for example, C: for your C drive. Alternatively you can switch back to the Classic menu of earlier versions of IE (see Figure 1-2). To do this you need to go to the Tools menu, select the Toolbars menu option, then ensure the Classic Menu option is selected.



Figure 1-2

The `<script>` Tag and Your First Simple JavaScript Program

Enough talk about the subject of JavaScript; it's time to look at how to put it into your web page. In this section, you write your first piece of JavaScript code.

Inserting JavaScript into a web page is much like inserting any other HTML content; you use tags to mark the start and end of your script code. The tag used to do this is the `<script>` tag. This tells the browser that the following chunk of text, bounded by the closing `</script>` tag, is not HTML to be displayed, but rather script code to be processed. The chunk of code surrounded by the `<script>` and `</script>` tags is called a *script block*.

Basically, when the browser spots `<script>` tags, instead of trying to display the contained text to the user, it uses the browser's built-in JavaScript interpreter to run the code's instructions. Of course, the code might give instructions about changes to the way the page is displayed or what is shown in the page, but the text of the code itself is never shown to the user.

You can put the `<script>` tags inside the header (between the `<head>` and `</head>` tags), or inside the body (between the `<body>` and `</body>` tags) of the HTML page. However, although you can put them outside these areas — for example, before the `<html>` tag or after the `</html>` tag — this is not permitted in the web standards and so is considered bad practice.

The `<script>` tag has a number of attributes, but the most important one is the `type` attribute. As you saw earlier, JavaScript is not the only scripting language available, and different scripting languages need to be processed in different ways. You need to tell the browser which scripting language to expect so that it knows how to process that language. Your opening script tag will look like this:

```
<script type="text/javascript">
```

Including the `type` attribute is good practice, but within a web page it can be left off. Browsers such as IE and Firefox use JavaScript as their default script language. This means that if the browser encounters a `<script>` tag with no `type` attribute set, it assumes that the script block is written in JavaScript. However, use of the `type` attribute is specified as mandatory by W3C (the World Wide Web Consortium), which sets the standards for HTML.

Okay, let's take a look at the first page containing JavaScript code.

Try It Out Painting the Document Red

This is a simple example of using JavaScript to change the background color of the browser. In your text editor (I'm using Windows Notepad), type the following:

```
<html>
<body BGCOLOR="WHITE">
<p>Paragraph 1</p>
<script type="text/javascript">
  document.bgColor = "RED";
</script>
</body>
</html>
```

Save the page as `ch1_examp1.htm` to a convenient place on your hard drive. Now load it into your web browser. You should see a red web page with the text `Paragraph 1` in the top-left corner. But wait — didn't you set the `<body>` tag's `BGCOLOR` attribute to white? Okay, let's look at what's going on here.

How It Works

The page is contained within `<html>` and `</html>` tags. This block contains a `<body>` element. When you define the opening `<body>` tag, you use HTML to set the page's background color to white.

```
<BODY BGCOLOR="WHITE">
```

Then you let the browser know that your next lines of code are JavaScript code by using the `<script>` start tag.

```
<script type="text/javascript">
```

Chapter 1: Introduction to JavaScript and the Web

Everything from here until the close tag, `</script>`, is JavaScript and is treated as such by the browser. Within this script block, you use JavaScript to set the document's background color to red.

```
document.bgColor = "RED";
```

What you might call the *page* is known as the *document* for the purpose of scripting in a web page. The document has lots of properties, including its background color, `bgColor`. You can reference properties of the `document` by writing `document`, followed by a dot, followed by the property name. Don't worry about the use of `document` at the moment; you look at it in depth later in the book.

Note that the preceding line of code is an example of a JavaScript *statement*. Every line of code between the `<script>` and `</script>` tags is called a statement, although some statements may run on to more than one line.

You'll also see that there's a semicolon (`;`) at the end of the line. You use a semicolon in JavaScript to indicate the end of a statement. In practice, JavaScript is very relaxed about the need for semicolons, and when you start a new line, JavaScript will usually be able to work out whether you mean to start a new line of code. However, for good coding practice, you should use a semicolon at the end of statements of code, and a single JavaScript statement should fit onto one line rather than continue on to two or more lines. Moreover, you'll find there are times when you must include a semicolon, which you'll come to later in the book.

Finally, to tell the browser to stop interpreting your text as JavaScript and start interpreting it as HTML, you use the script close tag:

```
</script>
```

You've now looked at how the code works, but you haven't looked at the order in which it works. When the browser loads in the web page, the browser goes through it, rendering it tag by tag from top to bottom of the page. This process is called *parsing*. The web browser starts at the top of the page and works its way down to the bottom of the page. The browser comes to the `<body>` tag first and sets the document's background to white. Then it continues parsing the page. When it comes to the JavaScript code, it is instructed to change the document's background to red.

Try It Out The Way Things Flow

Let's extend the previous example to demonstrate the parsing of a web page in action. Type the following into your text editor:

```
<html>
<body bgcolor="WHITE">
<p>Paragraph 1</p>
<script type="text/javascript">
  // Script block 1
  alert("First Script Block");
</script>
<p>Paragraph 2</p>
<script type="text/javascript">
  // Script block 2
```

```
document.bgColor = "RED";  
alert("Second Script Block");  
</script>  
<p>Paragraph 3</p>  
</body>  
</html>
```

Save the file to your hard drive as `ch1_examp2.htm` and then load it into your browser. When you load the page you should see the first paragraph, `Paragraph 1`, followed by a message box displayed by the first script block. The browser halts its parsing until you click the OK button. As you can see in Figure 1-3, the page background is white, as set in the `<body>` tag, and only the first paragraph is currently displayed.

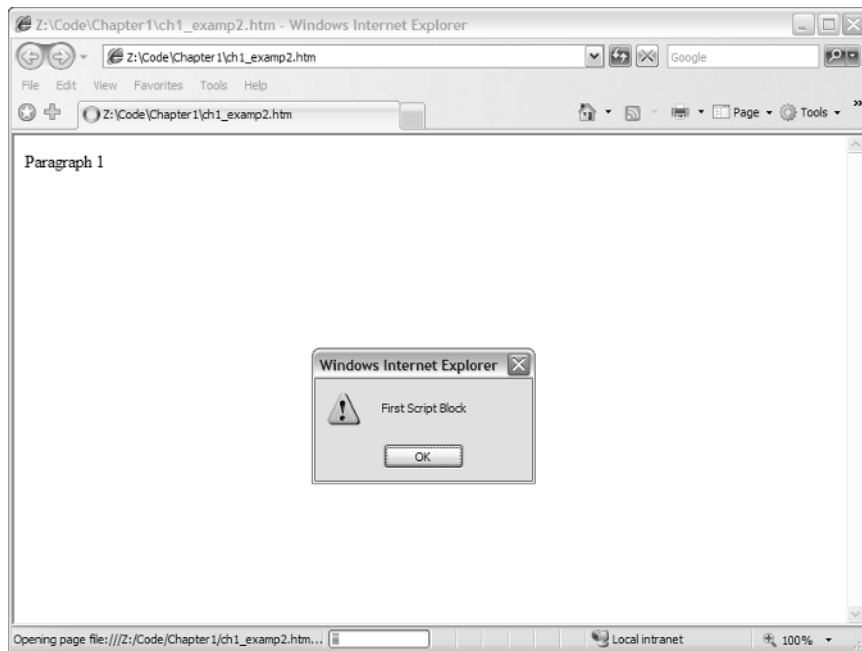


Figure 1-3

Click the OK button, and the parsing continues. The browser displays the second paragraph, and the second script block is reached, which changes the background color to red. Another message box is displayed by the second script block, as shown in Figure 1-4.

Click OK, and again the parsing continues, with the third paragraph, `Paragraph 3`, being displayed. The web page is complete, as shown in Figure 1-5.

Chapter 1: Introduction to JavaScript and the Web

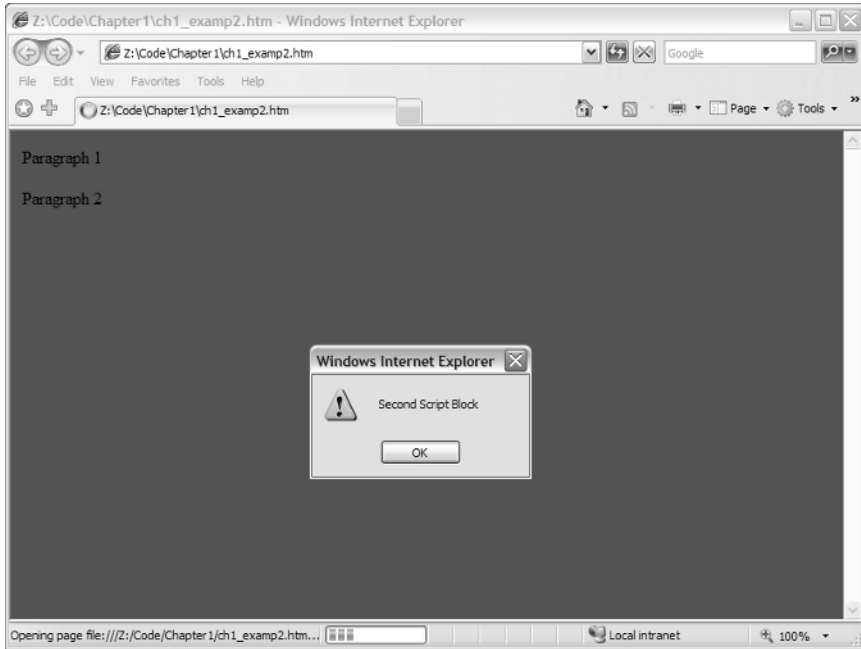


Figure 1-4

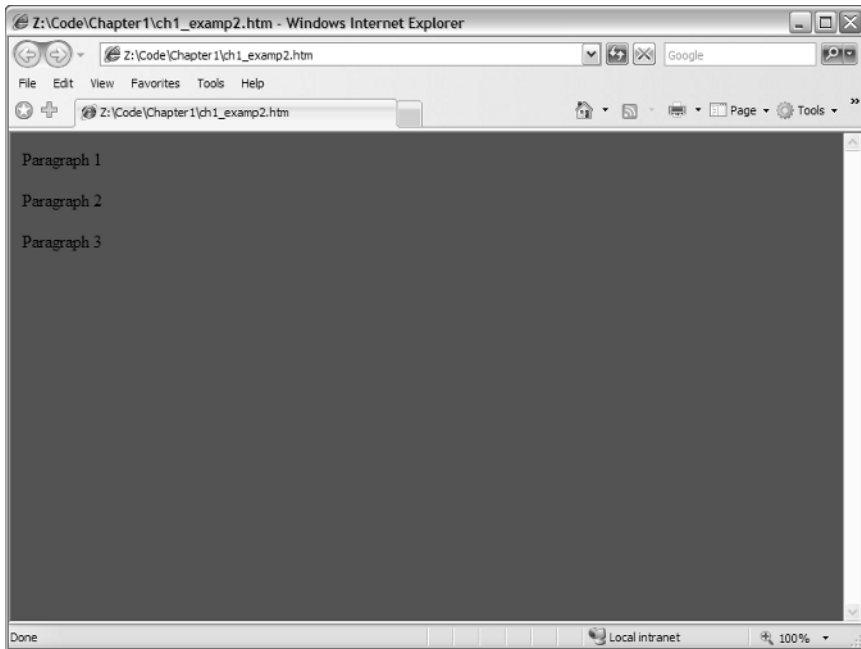


Figure 1-5

How It Works

The first part of the page is the same as in our earlier example. The background color for the page is set to white in the definition of the `<body>` tag, and then a paragraph is written to the page.

```
<html>
<body BGCOLOR="WHITE">
<p>Paragraph 1</p>
```

The first new section is contained in the first script block.

```
<script type="text/javascript">
  // Script block 1
  alert("First Script Block");
</script>
```

This script block contains two lines, both of which are new to you. The first line —

```
// Script block 1
```

is just a *comment*, solely for your benefit. The browser recognizes anything on a line after a double forward slash (`//`) to be a comment and does not do anything with it. It is useful for you as a programmer because you can add explanations to your code that make it easier to remember what you were doing when you come back to your code later.

The `alert()` function in the second line of code is also new to you. Before learning what it does, you need to know what a *function* is.

Functions are defined more fully in Chapter 3, but for now you need only to think of them as pieces of JavaScript code that you can use to do certain tasks. If you have a background in math, you may already have some idea of what a function is: A function takes some information, processes it, and gives you a result. A function makes life easier for you as a programmer because you don't have to think about how the function does the task — you can just concentrate on when you want the task done.

In particular, the `alert()` function enables you to alert or inform the user about something by displaying a message box. The message to be given in the message box is specified inside the parentheses of the `alert()` function and is known as the function's *parameter*.

The message box displayed by the `alert()` function is *modal*. This is an important concept, which you'll come across again. It simply means that the message box won't go away until the user closes it by clicking the OK button. In fact, parsing of the page stops at the line where the `alert()` function is used and doesn't restart until the user closes the message box. This is quite useful for this example, because it enables you to demonstrate the results of what has been parsed so far: The page color has been set to white, and the first paragraph has been displayed.

When you click OK, the browser carries on parsing down the page through the following lines:

```
<p>Paragraph 2</p>
<script type="text/javascript">
  // Script block 2
  document.bgColor = "RED";
  alert("Second Script Block");
</script>
```

Chapter 1: Introduction to JavaScript and the Web

The second paragraph is displayed, and the second block of JavaScript is run. The first line of the script block code is another comment, so the browser ignores this. You saw the second line of the script code in the previous example — it changes the background color of the page to red. The third line of code is the `alert()` function, which displays the second message box. Parsing is brought to a halt until you close the message box by clicking OK.

When you close the message box, the browser moves on to the next lines of code in the page, displaying the third paragraph and finally ending the web page.

```
<p>Paragraph 3</p>
</body>
</html>
```

Another important point raised by this example is the difference between setting properties of the page, such as background color, via HTML and doing the same thing using JavaScript. The method of setting properties using HTML is *static*: A value can be set only once and never changed again by means of HTML. Setting properties using JavaScript enables you to dynamically change their values. By the term *dynamic*, we are referring to something that can be changed and whose value or appearance is not set in stone.

This example is just that, an example. In practice if you wanted the page's background to be red, you would set the `<body>` tag's `BGColor` attribute to "RED", and not use JavaScript at all. Where you would want to use JavaScript is where you want to add some sort of intelligence or logic to the page. For example, if the user's screen resolution is particularly low, you might want to change what's displayed on the page; with JavaScript, you can do this. Another reason for using JavaScript to change properties might be for special effects — for example, making a page fade in from white to its final color.

A Brief Look at Browsers and Compatibility Problems

You've seen in the preceding example that by using JavaScript you can change a web page's document background color using the `bgColor` property of the `document`. The example worked whether you used a Netscape or Microsoft browser, because both types of browsers support a `document` with a `bgColor` property. You can say that the example is *cross-browser compatible*. However, it's not always the case that the property or language feature available in one browser will be available in another browser. This is even sometimes the case between versions of the same browser.

The version numbers for Internet Explorer and Firefox browsers are usually written as a decimal number; for example, Firefox has a version 1.5. In this book we use the following terminology to refer to these versions. By version 1.x we mean all versions starting with the number 1. By version 1.0+ we mean all versions with a number greater than or equal to 1.

One of the main headaches involved in creating web-based JavaScript is the differences between different web browsers, the level of HTML they support, and the functionality their JavaScript interpreters can handle. You'll find that in one browser you can move an image using just a couple of lines of code, but that in another it'll take a whole page of code, or even prove impossible. One version of JavaScript will contain a method to change text to uppercase, and another won't. Each new release of IE or Firefox browsers sees new and exciting features added to their HTML and JavaScript support. The good news is that to a much greater extent than ever before, browser creators are complying with standards set by organizations such as the W3C. Also, with a little ingenuity, you can write JavaScript that will work with both IE and Firefox browsers.

Which browsers you want to support really comes down to the browsers you think the majority of your web site's visitors, that is, your *user base*, will be using. This book is aimed at both IE6 and later and Firefox 1.5 and later.

If you want your web site to be professional, you need to somehow deal with older browsers. You could make sure your code is backward compatible—that is, it only uses features available in older browsers. However, you may decide that it's simply not worth limiting yourself to the features of older browsers. In this case you need to make sure your pages degrade gracefully. In other words, make sure that although your pages won't work in older browsers, they will fail in a way that means the user is either never aware of the failure or is alerted to the fact that certain features on the web site are not compatible with his or her browser. The alternative to degrading gracefully is for your code to raise lots of error messages, cause strange results to be displayed on the page, and generally make you look like an idiot who doesn't know what you're doing!

So how do you make your web pages degrade gracefully? You can do this by using JavaScript to determine which browser the web page is running in after it has been partially or completely loaded. You can use this information to determine what scripts to run or even to redirect the user to another page written to make best use of her particular browser. In later chapters, you see how to find out what features the browser supports and take appropriate action so that your pages work acceptably on as many browsers as possible.

Introducing the “Who Wants To Be A Billionaire” Trivia Quiz

Over the course of the first nine chapters of this book, you'll be developing a full web-based application, namely a trivia quiz. The trivia quiz works with both Firefox and IE6+ web browsers, making full use of their JavaScript capabilities.

Let's take a look at what the quiz will finally look like. The main starting screen is shown in Figure 1-6. Here the user can choose a time limit. Using a JavaScript-based timer, you keep track of how much time has elapsed.

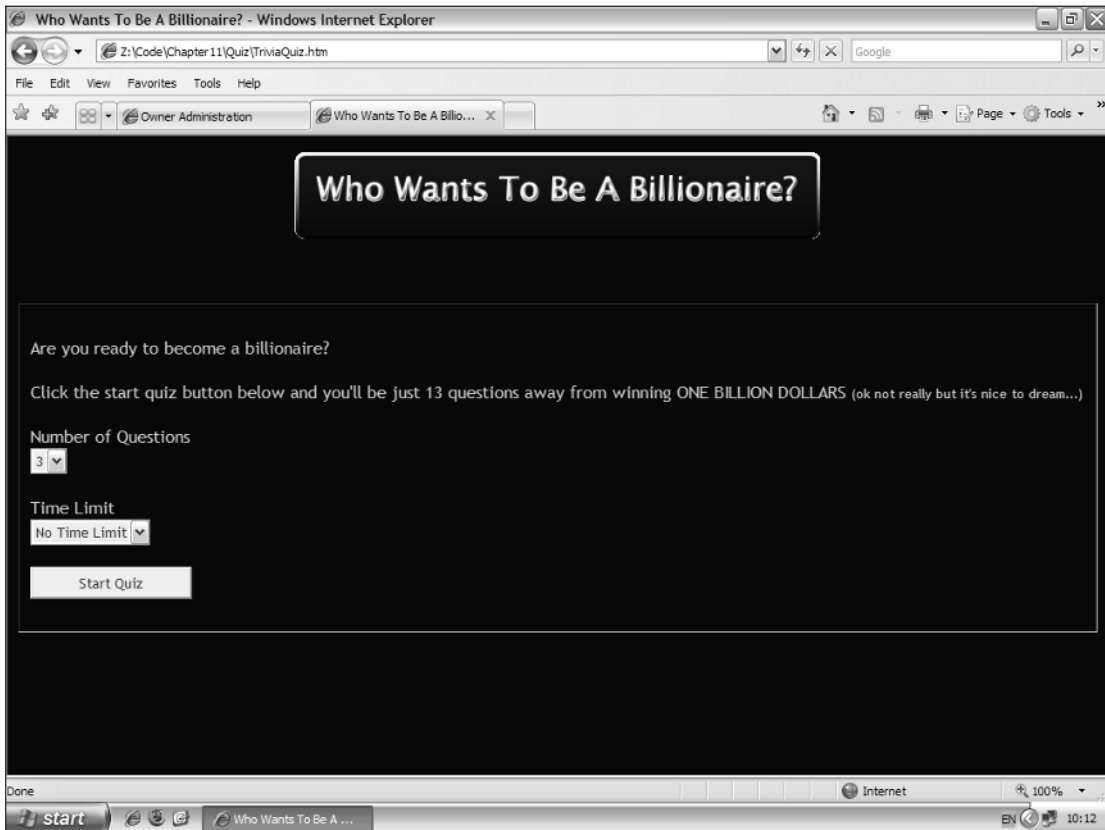


Figure 1-6

After clicking the Start Quiz button, the user is faced with a random choice of questions pulled from a database that you'll create to hold the trivia questions. There are two types of questions. The first, as shown in Figure 1-7, is the multiple-choice question. There is no limit to the number of answer options that you can specify for these types of questions: JavaScript handles them without the need for each question to be programmed differently.

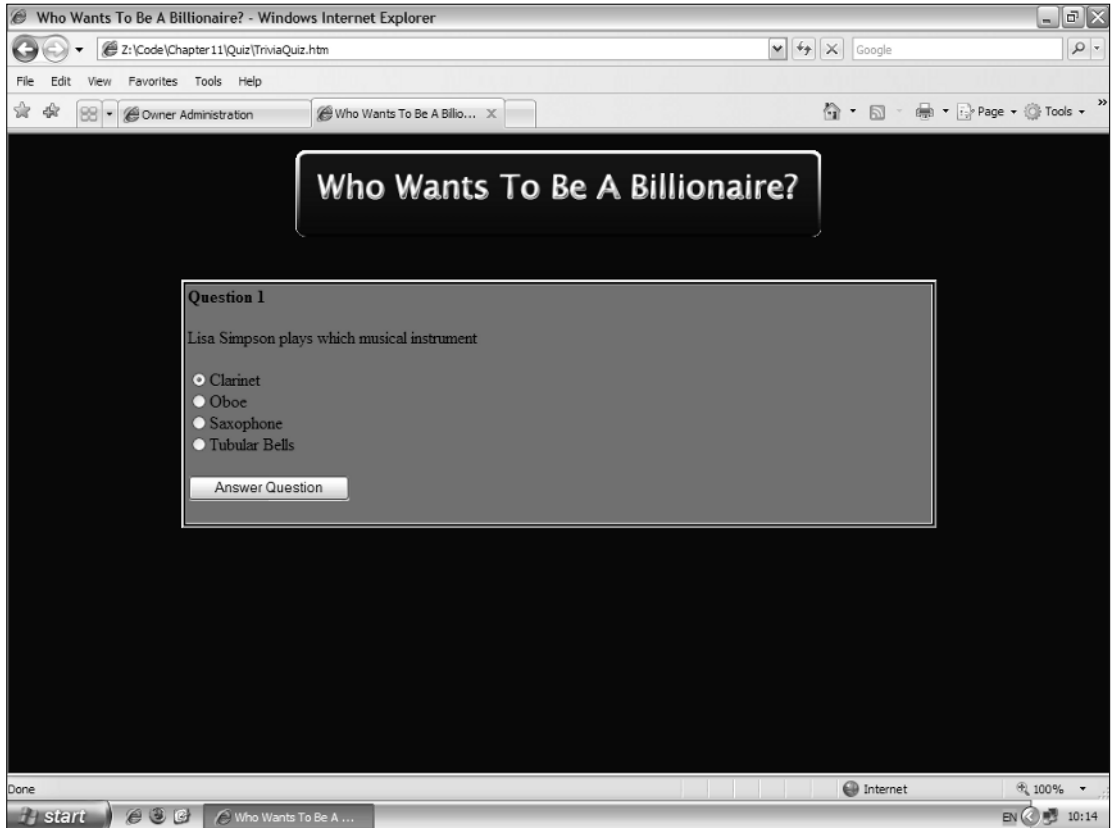


Figure 1-7

The second question style is a text-based one. The user types the answer into the text box provided, and then JavaScript does its best to intelligently interpret what the user has written. For example, for the question shown in Figure 1-8, Saxophone is the correct answer. However, the JavaScript has been programmed to also accept the abbreviated form sax as a correct answer. You find out how to do this in Chapter 8.

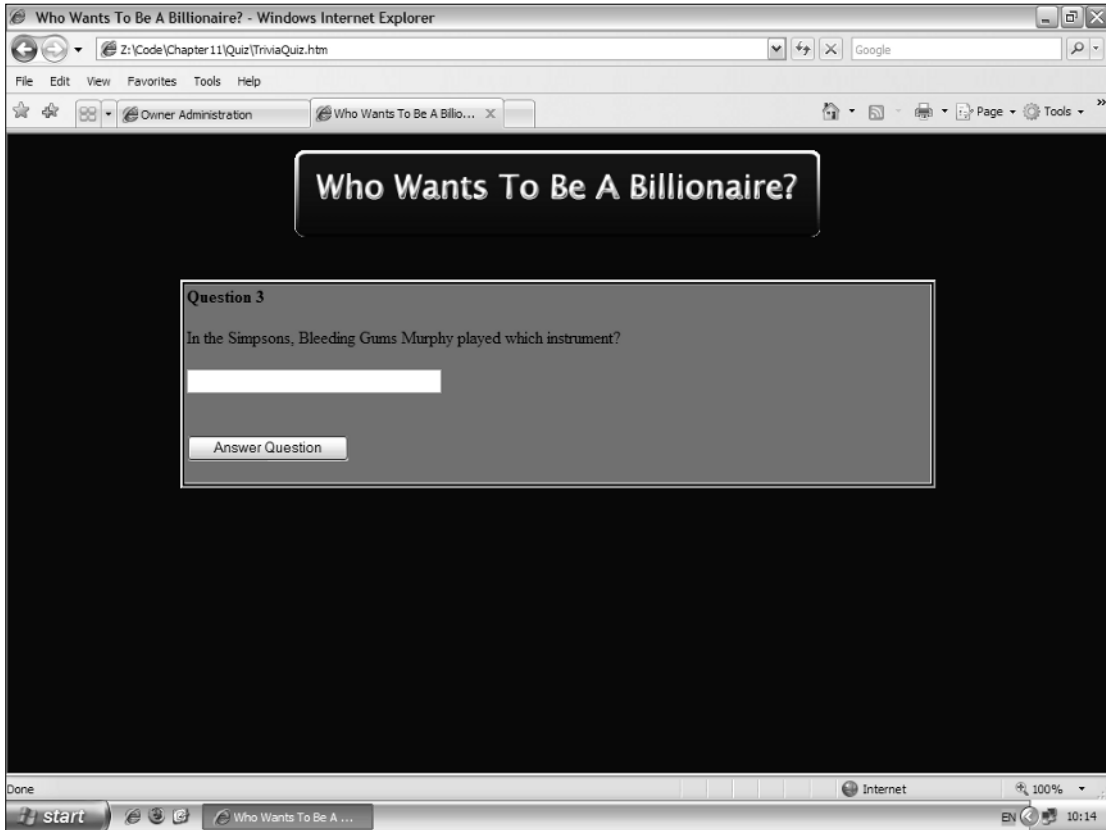


Figure 1-8

Finally, after the questions have all been answered, the final page of the quiz displays how much the user has won.

Ideas Behind the Coding of the Trivia Quiz

You've taken a brief look at the final version of the trivia quiz in action and will be looking at the actual code in later chapters, but it's worthwhile to consider the guiding principles behind its design and programming.

One of the most important ideas is code reuse. You save time and effort by making use of the same code again and again. Quite often in a web application you'll find that you need to do the same thing over and over again. For example, you'll need to make repeated use of the code that checks whether a question has been answered correctly. You could make as many copies of the code as you need, and add this code to your page wherever you need it. However, this makes maintaining the code difficult, because if you need to correct an error or add a new feature, you will need to make the change to the code in lots of different places. Once the code for a web application grows from a few lines in one page to many lines over a number of pages, it's quite difficult to actually keep track of the places where you have copied the code. So, with reuse in mind, the trivia quiz keeps in one place all the important code that will need to be used a number of times.

The same goes for any data you use. For example, in the trivia quiz you keep track of the number of questions that have been answered, and update this information in as few places as possible.

Sometimes you have no choice but to put important code in more than one place—for example, when you need information that can only be obtained in a particular circumstance. However, if you can keep it in one place, you'll find doing so makes coding more efficient.

In the trivia quiz, I've also tried to split the code into specific *functions*. You will be looking at JavaScript functions in detail in Chapter 3. In the trivia quiz, the function that provides a randomly selected question for your web page to display is in one place, regardless of whether this is a multiple-choice question or a purely text-based question. By doing this, you're not only writing code just once, you're also making life easier for yourself by keeping code that provides the same service or function in one place. As you'll see later in the book, the code for creating these different question types is very different, but at least putting it in the same logical place makes it easy to find.

When creating your own web-based applications, you might find it useful to break the larger concept, here a trivia quiz, into smaller ideas. Breaking it down makes writing the code a lot easier. Rather than sitting down with a blank screen and thinking, "Right, now I must write a trivia quiz," you can think, "Right, now I must write some code to create a question." We find this technique makes coding a lot less scary and easier to get started on. This method of splitting the requirements of a piece of code into smaller and more manageable parts is often referred to as "divide and conquer."

Let's use the trivia quiz as an example. The trivia quiz application needs to do the following things:

- Ask a question.
- Retrieve and check the answer provided by the user to see if it's correct.
- Keep track of how many questions have been asked.
- Keep track of how many questions the user has answered correctly.
- If it's a timed quiz, keep track of the time remaining, and stop the quiz when the time is up.
- Show a final summary of the number of correct answers given out of the number answered.

These are the core ingredients for the trivia quiz. You may want to do other things, such as keep track of the number of user visits, but these are really external to the functionality of the trivia quiz.

After you've broken the whole concept into various logical areas, it's sometimes worth using the divide-and-conquer technique again to break the areas down into even smaller chunks, particularly if an area is quite complex or involved. As an example, let's take the first item from the preceding list.

Asking a question will involve the following:

- Retrieving the question data from where they're stored, for example from a database.
- Processing the data and converting them into a form that can be presented to the user. Here you need to create HTML to be displayed in a web page. How the data are processed depends on the question style: multiple choice or text based.
- Displaying the question for the user to answer.

Chapter 1: Introduction to JavaScript and the Web

As you build up the trivia quiz over the course of the book, you'll look at its design and some of the tricks and tactics that go into that design in more depth. You'll also break down each function as you come to it, to make it clear what needs to be done.

What Functionality to Add and Where?

How do you build up the functionality needed in the trivia quiz? The following list should give you an idea of what you add and in which chapter.

In Chapter 2, you start the quiz off by defining the multiple-choice questions that will be asked. You do this using something called an *array*, which is also introduced in that chapter.

In Chapter 3, where you learn about functions in more detail, you add a function to the code that will check to see whether the user has entered the correct answer.

After a couple of chapters of theory, in Chapter 6 you get the quiz into its first “usable” state. You display the questions to the user, and allow the user to answer those questions.

In Chapter 7, you enhance the quiz by turning it into what is called a *multi-frame application*. You add a button that the user can click to start the quiz, and specify that the quiz must finish after all the questions have been asked, repeating them indefinitely.

In Chapter 8, you add the text-based questions to the quiz. These must be treated slightly differently from multiple-choice questions, both in how they are displayed to the user and in how the user's answers are checked. As you saw earlier, the quiz will accept a number of different correct answers for these questions.

In Chapter 9, you allow the user to choose whether he or she wants to have a time limit for the quiz. If users choose to impose a time limit upon themselves, you count down the time in the status bar of the window and inform them when their time is up.

In Chapter 11, you complete the quiz by storing information about the user's previous results, using *cookies*, which are introduced in that chapter. This enables us to give the user a running average score at the end of the quiz.

Summary

At this point you should have a feel for what JavaScript is and what it can do. In particular, this brief introduction covered the following:

- ❑ You looked into the process the browser follows when interpreting your web page. It goes through the page element by element (parsing), and acts upon your HTML tags and JavaScript code as it comes to them.
- ❑ When you are developing for the web using JavaScript, you can choose to have your code executed in one of two places: server-side or client-side. Client-side is essentially the side on which the browser is running — the user's machine. Server-side refers to any processing or storage done on the web server itself.

- ❑ Unlike many programming languages, JavaScript requires just a text editor to start creating code. Something like Windows Notepad is fine for getting started, though more extensive tools will prove valuable once you get more experience.
- ❑ JavaScript code is embedded into the web page itself along with the HTML. Its existence is marked out by the use of `<script>` tags. As with HTML, script executes from the top of the page and works down to the bottom, interpreting and executing the code statement by statement.
- ❑ You were introduced to the online trivia quiz, which is the case study that you'll be building over the course of the book. You took a look at some of the design ideas behind the trivia quiz's coding, and learned how the functionality of the quiz will be built up over the course of the book.

