THE BOOK of

JAVASCRIPT

2ND EDITION

A PRACTICAL GUIDE TO *INTERACTIVE* WEB PAGES

thau!

NOW WITH Ajax!

NO STARCH PRESS

2

USING VARIABLES AND BUILT-IN FUNCTIONS TO UPDATE YOUR WEB PAGES AUTOMATICALLY

With JavaScript you can update the content on your pages automatically—every day, every hour, or every second. In this chapter, I'll focus on a simple script that automatically changes the date on your web page.

Along the way you'll learn:

- How JavaScript uses variables to remember simple items such as names and numbers
- How JavaScript keeps track of more complicated items such as dates
- How to use JavaScript functions to write information to your web page

Before getting into the nuts and bolts of functions and variables, let's take a look at a couple of examples of web pages that automatically update themselves, starting with the European Space Agency (http://www.esa.int). As you can see in Figure 2-1, the ESA's home page shows you the current date. Rather than change the home page every day, the ESA uses JavaScript to change the date automatically.

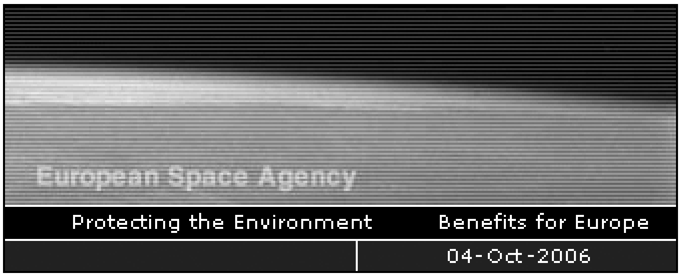


Figure 2-1: Using JavaScript to display the current date

An even more frequently updated page is the home page of the *Book of JavaScript* website (http://www.bookofjavascript.com), which updates the time as well as the date (see Figure 2-2). You don't have to sit in front of your computer, updating the dates and times on your websites. JavaScript can set you free! The ability to write HTML to web pages dynamically is one of JavaScript's most powerful features.

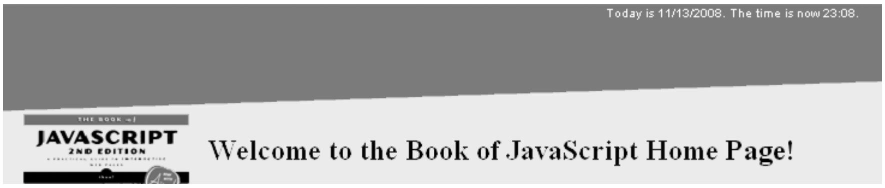

Today is 11/13/2008. The time is now 23:08.
THE BOOK of JAVASCRIPT 2ND EDITION
Welcome to the Book of JavaScript Home Page!

Figure 2-2: Dynamically updating the date and time

To understand how to update the date and time on the page, you'll first have to learn about variables, strings, and functions. Your homework assignment at the end of this chapter will be to figure out how to add seconds to the time.

Variables Store Information

Think back to those glorious days of algebra class when you learned about variables and equations. For example, if $x = 2$, $y = 3$, and $z = x + y$, then $z = 5$. In algebra, variables like x, y, and z store or hold the place of numbers. In JavaScript and other programming languages, variables also store other kinds of information.

Syntax of Variables

The *syntax* of variables (the rules for defining and using variables) is slightly different in JavaScript from what it was in your algebra class. Figure 2-3 illustrates the syntax of variables in JavaScript with a silly script that figures out how many seconds there are in a day.

NOTE *Figure 2-3 does not write the results of the JavaScript to the web page—I'll explain how to do that in Figure 2-4.*

```
  <html>
  <head>
  <title>Seconds in a Day</title>

  <script type = "text/javascript">
  <!-- hide me from older browsers

❶ var seconds_per_minute = 60;
  var minutes_per_hour = 60;
  var hours_per_day = 24;

❷ var seconds_per_day = seconds_per_minute * minutes_per_hour * hours_per_day;

  // show me -->
❸ </script>
  </head>
  <body>

  <h1>Know how many seconds are in a day?</h1>
  <h2>I do!</h2>

  </body>
  </html>
```

Figure 2-3: Defining and using variables

There's a lot going on here, so let's take it line by line. Line ❶ is a *statement* (a statement in JavaScript is like a sentence in English), and it says to JavaScript, "Create a variable called `seconds_per_minute` and set its value to 60." Notice that ❶ ends with a semicolon. Semicolons in JavaScript are like periods in English: They mark the end of a statement (for example, one that defines a variable, as above). As you see more and more statements, you'll get the hang of where to place semicolons.

The first word, `var`, introduces a variable for the first time—you don't need to use it after the first instance, no matter how many times you employ the variable in the script.

NOTE *Many people don't use* `var` *in their code. Although most browsers let you get away without it, it's always a good idea to put* `var` *in front of a variable the first time you use it. (You'll see why when I talk about writing your own functions in Chapter 6.)*

Naming Variables

Notice that the variable name in ❶ is pretty long—unlike algebraic variables, it's not just a single letter like *x*, *y*, or *z*. When using variables in JavaScript (or any programming language), you should give them names that indicate what piece of information they hold. The variable in ❶ stores the number of seconds in a minute, so I've called it `seconds_per_minute`.

If you name your variables descriptively, your code will be easier to understand while you're writing it, and much easier to understand when you return to it later for revision or enhancement. Also, no matter which programming language you use, you'll spend about 50 percent of your coding time finding and getting rid of your mistakes. This is called *debugging*—and it's a lot easier to debug code when the variables have descriptive names. You'll learn more about debugging in Chapter 14.

There are four rules for naming variables in JavaScript:

1. The initial character must be a letter, an underscore, or a dollar sign, but subsequent characters may be numbers as well.
2. No spaces are allowed.
3. Variables are case sensitive, so `my_cat` is different from `My_Cat`, which in turn is different from `mY_cAt`. As far as the computer is concerned, each of these would represent a different variable—even if that's not what the programmer intended. (You'll see an example of this in the section "alert()" on page 22.) To avoid any potential problems with capitalization, I use lowercase for all my variables, with underscores (_) where there would be spaces in ordinary English.
4. You can't use reserved words. *Reserved words* are terms used by the JavaScript language itself. For instance, you've seen that the first time you use a variable, you should precede it with the word `var`. Because JavaScript uses the word `var` to introduce variables, you can't use `var` as a variable name. Different browsers have different reserved words, so the best thing to do is avoid naming variables with words that seem like terms JavaScript might use. Most reserved words are fairly short, so using longer, descriptive variable names keeps you fairly safe. I often call my variables things like `the_cat`, or `the_date` because there are no reserved words that start with the word *the*. If you have a JavaScript that you're *certain* is correct, but it isn't working for some reason, it might be because you've used a reserved word.

Arithmetic with Variables

Line ❷ in Figure 2-3 introduces a new variable called `seconds_per_day` and sets it equal to the product of the other three variables using an asterisk (*), which means multiplication. A plus sign (+) for addition, a minus sign (-) for subtraction, and a slash (/) for division represent the other major arithmetic functions.

When the browser finishes its calculations in our example, it reaches the end of the JavaScript in the head (❸) and goes down to the body of the HTML. There it sees two lines of HTML announcing that the page knows how many seconds there are in a day.

```
<h1>Know how many seconds are in a day?</h1>
<h2>I do!</h2>
```

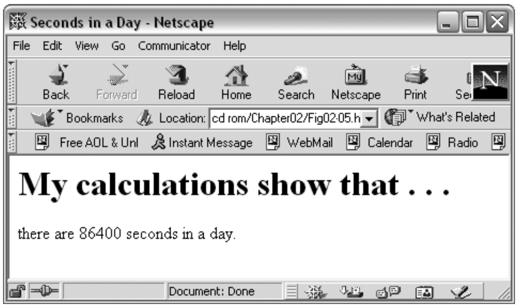
So now you have a page that knows how many seconds there are in a day. Big deal, right? Wouldn't it be better if you could tell your visitors what the answer is? Well, you can, and it's not very hard.

Write Here Right Now: Displaying Results

JavaScript uses the `write()` function to write text to a web page. Figure 2-4 shows how to use `write()` to let your visitors know how many seconds there are in a day. (The new code is in bold.) Figure 2-5 shows the page this code displays.

```
<html>
<head>
<title>Seconds in a Day</title>

<script type = "text/javascript">
<!-- hide me from older browsers

var seconds_per_minute = 60;
var minutes_per_hour = 60;
var hours_per_day = 24;

var seconds_per_day = seconds_per_minute * minutes_per_hour * hours_per_day;

// show me -->
</script>
</head>
<body>

<h1>My calculations show that . . .</h1>

<script type = "text/javascript">
<!-- hide me from older browsers

❶ window.document.write("there are ");
window.document.write(seconds_per_day);
window.document.write(" seconds in a day.");

// show me -->
</script>

</body>
</html>
```

Figure 2-4: Using `write()` to write to a web page


Seconds in a Day - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Se
Bookmarks Location: cd rom/Chapter02/Fig02-05.h What's Related
Free AOL & Unl Instant Message WebMail Calendar Radio
Document: Done

My calculations show that . . .

there are 86400 seconds in a day.

Figure 2-5: JavaScript's calculations

Line-by-Line Analysis of Figure 2-4

Line ❶ in Figure 2-4 writes the words *there are* to the web page (only the words between the quote marks appear on the page). Don't worry about all the periods and what `window` and `document` really mean right now (I'll cover these topics in depth in Chapter 4, when we talk about image swaps). For now, just remember that if you want to write something to a web page, use `window.document.write("whatever");`, placing the text you want written to the page between the quotes. If you don't use quotes around your text, as in

```
window.document.write(seconds_per_day);
```

then JavaScript interprets the text between the parentheses as a variable and writes whatever is stored in the variable (in this case, `seconds_per_day`) to the web page (see Figure 2-6). If you accidentally ask JavaScript to write out a variable you haven't defined, you'll get a JavaScript error.

Be careful not to put quotes around variable names if you want JavaScript to know you're talking about a variable. If you add quotes around the `seconds_per_day` variable, like this:

```
window.document.write("seconds_per_day");
```

then JavaScript will write *seconds_per_day* to the web page. The way JavaScript knows the difference between variables and regular text is that regular text has quotes around it and a variable doesn't.

Strings

Any series of characters between quotes is called a *string*. (You'll be seeing lots of strings throughout this book.) Strings are a basic type of information, like numbers—and like numbers, you can assign them to variables.

To assign a string to a variable, you'd write something like this:

```
var my_name = "thau!";
```

The word `thau!` is the string assigned to the variable `my_name`.

You can stick strings together with a plus sign (+), as shown in the bolded section of Figure 2-6. This code demonstrates how to write output to your page using strings.

```
<html>
<head>
<title>Seconds in a Day</title>
<script type = "text/javascript">
<!-- hide me from older browsers

var seconds_per_minute = 60;
var minutes_per_hour = 60;
var hours_per_day = 24;

var seconds_per_day = seconds_per_minute * minutes_per_hour * hours_per_day;
```

```
   // show me -->
   </script>
   </head>
   <body>

   <h1>My calculations show that . . .</h1>

   <script type = "text/javascript">
   <!-- hide me from older browsers

❶ var first_part = "there are ";
❷ var last_part = " seconds in a day.";
❸ var whole_thing = first_part + seconds_per_day + last_part;

   window.document.write(whole_thing);

   // show me -->
   </script>

   </body>
   </html>
```

Figure 2-6: Putting strings together

Line-by-Line Analysis of Figure 2-6

Line ❶ in Figure 2-6,

```
var first_part = "there are ";
```

assigns the string `"there are"` to the variable `first_part`. Line ❷,

```
var last_part = " seconds in a day.";
```

sets the variable `last_part` to the string `"seconds in a day."` Line ❸ glues together the values stored in `first_part`, `seconds_per_day`, and `last_part`. The end result is that the variable `whole_thing` includes the whole string you want to print to the page, *there are 86400 seconds in a day*. The `window.document.write()` line then writes `whole_thing` to the web page.

NOTE *The methods shown in Figures 2-4 and 2-6 are equally acceptable ways of writing* there are 86400 seconds in a day. *However, there are times when storing strings in variables and then assembling them with the plus sign (+) is clearly the best way to go. We'll see a case of this when we finally get to putting the date on a page.*

More About Functions

Whereas variables store information, *functions* process that information.

All functions take the form *`functionName`*`()`. Sometimes there's something in the parentheses and sometimes there isn't. You've already seen one of JavaScript's many built-in functions, `window.document.write()`, which writes whatever lies between the parentheses to the web page. Before diving into the date functions that you'll need to write the date to your web page, I'll talk about two interesting functions, just so you get the hang of how functions work.

alert()

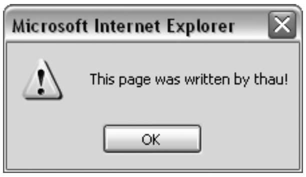
One handy function is `alert()`, which puts a string into a little announcement box (also called an *alert box*). Figure 2-7 demonstrates how to call an `alert()`, and Figure 2-8 shows what the alert box looks like.

```
  <html>
  <head>
  <title>An Alert Box</title>

  <script type = "text/javascript">
  <!-- hide me from older browsers
❶ alert("This page was written by thau!");
  // show me -->
  </script>

  <body>
❷ <h1>To code, perchance to function</h1>
  </body>
  </html>
```

Figure 2-7: Creating an alert box

The first thing visitors see when they come to the page Figure 2-7 creates is an alert box announcing that I wrote the page (Figure 2-8). The alert box appears because of ❶, which tells JavaScript to execute its `alert()` function.

While the alert box is on the screen, the browser stops doing any work. Clicking OK in the alert box makes it go away and allows the browser to finish drawing the web page. In this case, that means writing the words *To code, perchance to function* to the page (❷).


Microsoft Internet Explorer

This page was written by thau!

Figure 2-8: The alert box

The `alert()` function is useful for troubleshooting when your JavaScript isn't working correctly. Let's say you've typed in Figure 2-6, but when you run the code, you see that you must have made a typo—it says there are 0 seconds in a day instead of 86400. You can use `alert()` to find out how the different variables are set before multiplication occurs. The script in Figure 2-9 contains an error that causes the script to say there are "undefined" seconds in a year; and to track down the error, I've added `alert()` function statements that tell you why this problem is occurring.

```
<html>
<head>
<title>Seconds in a Day</title>

<script type = "text/javascript">
<!-- hide me from older browsers

var seconds_per_minute = 60;
var minutes_per_hour = 60;
❶ var Hours_per_day = 24;
❷ alert("seconds per minute is: " + seconds_per_minute);
❸ alert("minutes per hour is: " + minutes_per_hour);
❹ alert("hours per day is: " + hours_per_day);
❺ var seconds_per_day = seconds_per_minute * minutes_per_hour * hours_per_day;



// show me -->
</script>
</head>
<body>

<h1>My calculations show that . . .</h1>

<script type = "text/javascript">
<!-- hide me from older browsers

var first_part = "there are ";
var last_part = " seconds in a day.";
var whole_thing = first_part + seconds_per_day + last_part;

window.document.write(whole_thing);

// show me -->
</script>

</body>
</html>
```

Figure 2-9: Using `alert()` to find out what's wrong

Line-by-Line Analysis of Figure 2-9

The problem with this script is in ❶. Notice the accidental capitalization of the first letter in `Hours_per_day`. This is what causes the script to misbehave. Line ❺ multiplies the other numbers by the variable `hours_per_day`, but `hours_per_day` was not set—remember, JavaScript considers it a different variable from `Hours_per_day`—so JavaScript thinks its value is either 0 or undefined, depending on your browser. Multiplying anything by 0 results in 0, so the script calculates that there are 0 seconds in a day. The same holds true for browsers that think `hours_per_day` is undefined. Multiplying anything by something undefined results in the answer being undefined, so the browser will report that there are undefined seconds in a day.

This script is short, making it easy to see the mistake. However, in longer scripts it's sometimes hard to figure out what's wrong. I've added ❷, ❸, and ❹ in this example to help diagnose the problem. Each of these statements puts a variable into an alert box. The alert on ❷ will say `seconds_per_minute is: 60`. The alert on ❹ will say `hours_per_day is: 0`, or, depending on your browser, the alert won't appear at all. Either way, you'll know there's a problem with the `hours_per_day` variable. If you can't figure out the mistake by reading the script, you'll find this type of information very valuable. Alerts are very useful debugging tools.

prompt()

Another helpful built-in function is `prompt()`, which asks your visitor for some information and then sets a variable equal to whatever your visitor types. Figure 2-10 shows how you might use `prompt()` to write a form letter.

```
  <html>
  <head>
  <title>A Form Letter</title>
  <script type = "text/javascript">
  <!-- hide me from older browsers

❶ var the_name = prompt("What's your name?", "put your name here");

  // show me -->
  </script>
  </head>
  <body>
❷ <h1>Dear

  <script type = "text/javascript">
  <!-- hide me from older browsers

  document.write(the_name);

  // show me -->
  </script>

  ,</h1>

  Thank you for coming to my web page.

  </body>
  </html>
```

Figure 2-10: Using `prompt()` to write a form letter

Notice that `prompt()` in ❶ has two strings inside the parentheses: `"What's your name?"` and `"put your name here"`. If you run the code in Figure 2-10, you'll see a prompt box that resembles Figure 2-11. (I've used the Opera browser in this illustration; prompt boxes will look somewhat different in IE and other browsers.) If you type Rumpelstiltskin and click OK, the page responds with *Dear Rumpelstiltskin, Thank you for coming to my web page.*

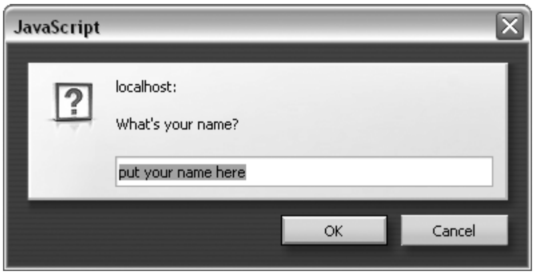


Figure 2-11: Starting a form letter with a prompt box

The text above the box where your visitors will type their name ("What's your name?") is the first string in the prompt function; the text inside the box ("put your name here") is the second string. If you don't want anything inside the box, put two quotes ("") right next to each other in place of the second string to keep that space blank:

```
var the_name = prompt("What's your name?", "");
```

If you look at the JavaScript in the body (starting in ❷), you'll see how to use the variable the_name. First write the beginning of the heading to the page using normal HTML. Then launch into JavaScript and use document.write(the_name) to write whatever name the visitor typed into the prompt box for your page. If your visitor typed yertle the turtle into that box, *yertle the turtle* gets written to the page. Once the item in the_name is written, you close the JavaScript tag, write a comma and the rest of the heading using regular old HTML, and then continue with the form letter. Nifty, eh?

The prompt() function is handy because it enables your visitor to supply the variable information. In this case, after the user types a name into the prompt box in Figure 2-10 (thereby setting the variable the_name), your script can use the supplied information by calling that variable.

Parameters

The words inside the parentheses of functions are called *parameters*. The document.write() function requires one parameter: a string to write to your web page. The prompt() function takes two parameters: a string to write above the box and a string to write inside the box.

Parameters are the only aspect of a function you can control; they are your means of providing the function with the information it needs to do its job. With a prompt() function, for example, you can't change the color of the box, how many buttons it has, or anything else; in using a predefined prompt box, you've decided that you don't need to customize the box's appearance. You can only change the parameters it specifically provides—

namely, the text and heading of the prompt you want to display. You'll learn more about controlling what functions do when you write your own functions in Chapter 6.

Writing the Date to Your Web Page

Now that you know about variables and functions, you can print the date to your web page. To do so, you must first ask JavaScript to check the local time on your visitor's computer clock:

```
var now = new Date();
```

The first part of this line, `var now =`, should look familiar. It sets the variable `now` to some value. The second part, `new Date()`, is new; it creates an object.

Objects store data that require multiple pieces of information, such as a particular moment in time. For example, in JavaScript you need an object to describe *2:30 PM on Saturday, January 7, 2006, in San Francisco*. That's because it requires many different bits of information: the time, day, month, date, and year, as well as some representation (in relation to Greenwich Mean Time) of the user's local time. As you can imagine, working with an object is a bit more complicated than working with just a number or a string.

Because dates are so rich in information, JavaScript has a built-in `Date` object to contain those details. When you want the user's current date and time, you use `new Date()` to tell JavaScript to create a `Date` object with all the correct information.

NOTE *You must capitalize the letter* `D` *in* `Date` *to tell JavaScript you want to use the built-in* `Date` *object. If you don't capitalize it, JavaScript won't know what kind of object you're trying to create, and you'll get an error message.*

Built-in Date Functions

Now that JavaScript has created your `Date` object, let's extract information from it using JavaScript's built-in date functions. To extract the current year, use the `Date` object's `getYear()` function:

```
var now = new Date();
var the_year = now.getYear();
```

Date and Time Methods

In the code above, the variable `now` is a `Date` object, and the function `getYear()` is a method of the `Date` object. *Methods* are simply functions that are built in to objects. For example, the `getYear()` function is built in to the `Date` object and gets the object's year. Because the function is part of the `Date` object, it is called a method. To use the `getYear()` method to get the year of the date stored in the variable `now`, you would write:

```
now.getYear()
```

Table 2-1 lists commonly used date methods. (You can find a complete list of date methods in Appendix C.)

Table 2-1: Commonly Used Date and Time Methods

Name	Description
`getDate()`	The day of the month as an integer from 1 to 31
`getDay()`	The day of the week as an integer where 0 is Sunday and 1 is Monday
`getHours()`	The hour as an integer between 0 and 23
`getMinutes()`	The minutes as an integer between 0 and 59
`getMonth()`	The month as an integer between 0 and 11 where 0 is January and 11 is December
`getSeconds()`	The seconds as an integer between 0 and 59
`getTime()`	The current time in milliseconds where 0 is January 1, 1970, 00:00:00
`getYear()`	The year, but this format differs from browser to browser

NOTE *Notice that `getMonth()` returns a number between 0 and 11; if you want to show the month to your site's visitors, to be user-friendly you should add 1 to the month after using `getMonth()` as shown in ❷ in Figure 2-12.*

Internet Explorer and various versions of Netscape deal with years in different and strange ways:

- Some versions of Netscape, such as Netscape 4.0 for the Mac, always return the current year minus 1900. So if it's the year 2010, `getYear()` returns 110.
- Other versions of Netscape return the full four-digit year except when the year is in the twentieth century, in which case they return just the last two digits.
- Netscape 2.0 can't deal with dates before 1970 at all. Any date before January 1, 1970 is stored as December 31, 1969.
- In Internet Explorer, `getYear()` returns the full four-digit year if the year is after 1999 or before 1900. If the year is between 1900 and 1999, it returns the last two digits.

You'd figure a language created in 1995 wouldn't have the Y2K problem, but the ways of software developers are strange. Later in this chapter I'll show you how to fix this bug.

Code for Writing the Date and Time

Now let's put this all together. To get the day, month, and year, we use the `getDate()`, `getMonth()`, and `getYear()` methods. To get the hour and the minutes, we use `getHours()` and `getMinutes()`.

Figure 2-12 shows you the complete code for writing the date and time (without seconds) to a web page, as seen on the *Book of JavaScript* home page.

```
  <html>
  <head><title>The Book of JavaScript</title>
  <script type = "text/javascript">
  <!-- hide me from older browsers
  // get the Date object
  //
❶ var date = new Date();

  // get the information out of the Date object
  //
  var month = date.getMonth();
  var day = date.getDate();
  var year = date.getYear();
  var hour = date.getHours();
  var minutes = date.getMinutes();
❷ month = month + 1;  // because January is month 0
  // fix the Y2K bug
  //
❸ year = fixY2K(year);

  // fix the minutes by adding a 0 in front if it's less than 10
  //
❹ minutes = fixTime(minutes);

  // create the date string
  //
❺ var date_string = month + "/" + day + "/" + year;
❻ var time_string = hour + ":" + minutes;
❼ var date_time_string = "Today is " + date_string + ".  The time is now " +
      time_string + ".";

  // This is the Y2K fixer function--don't worry about how this works,
  // but if you want it in your scripts, you can cut and paste it.
  //
  function fixY2K(number) {
    if (number < 1000) {
      number = number + 1900;
    }
    return number;
  }

  // This is the time fixer function--don't worry about how this works either.
  function fixTime(number) {
    if (number < 10) {
      number = "0" + number;
    }
    return number;
  }

  // show me -->
  </script>
  </head>
  <body>
```

```
❽ <h1>Welcome to the Book of JavaScript Home Page!</h1>

  <script type = "text/javascript">
  <!-- hide me from older browsers
❾ document.write(date_time_string);
  // show me -->
  </script>
  </body>
  </html>
```

Figure 2-12: Writing the current date and time to a web page

Line-by-Line Analysis of Figure 2-12

Here are a few interesting things in this example.

Getting the Date and Time

The lines from ❶ up until ❷ get the current date and time from the visitor's computer clock and then use the appropriate date methods to extract the day, month, year, hours and minutes. Although I'm using a variable name `date` in ❶ to store the date, I could have used any variable name there: `the_date`, `this_moment`, `the_present`, or any valid variable name. Don't be fooled into thinking that a variable needs to have the same name as the corresponding JavaScript object; in this case, `date` just seems like a good name.

Making Minor Adjustments

Before building the strings we will write to the website, we need to make some little adjustments to the date information just collected. Here's how it works:

- Line ❷ adds 1 to the month because `getMonth()` thinks January is month 0.
- Line ❸ fixes the Y2K problem discussed earlier in the chapter, in which the `getYear()` method returns the wrong thing on some older browsers. If you feed `fixY2K()` the year returned by `date.getYear()`, it will return the correct year. The `fixY2K()` function is not a built-in JavaScript function. I had to write it myself. Don't worry about how the function works right now.
- Line ❹ fixes a minor formatting issue, using another function that's not built-in. If the script is called at 6 past the hour, `date.getMinutes()` returns 6. If you don't do something special with that 6, your time will look like 11:6 instead of 11:06. So `fixTime()` sticks a zero in front of a number if that number is less than 10. You can use `fixTime()` to fix the seconds too, for your homework assignment.

Getting the String Right

Now that we've made a few minor adjustments, it's time to build the strings. Line ❺ builds the string for the date. Here's the *wrong* way to do it:

```
var date_string = "month / day / year";
```

If you wrote your code this way, you'd get a line that says *Today is month / day / year*. Why? Remember that JavaScript doesn't look up variables if they're inside quotes. So place the variables outside the quote marks and glue everything together using plus signs (+):

```
var date_string = month + "/" + day + "/" + year;
```

This may look a little funny at first, but it's done so frequently that you'll soon grow used to it. Line ❻ creates the string to represent the time. It is very similar to ❺. Line ❼ puts ❺ and ❻ together to create the string that will be written to the website. Lines ❺ through ❼ could all have been written as one long line:

```
var date_time_string = "Today is " + month + "/" + day + "/" + year +
    ". The time is now " + hour + ":" + minutes + ".";
```

However, using three lines makes the code easier for people to read and understand. It's always best to write your code as if other people are going to read it.

What Are Those Other Functions?

The JavaScript between ❼ and ❽ defines the `fixY2K()` and `fixTime()` functions. Again, don't worry about these lines for now. We'll cover how to write your own functions in glorious detail in Chapter 6.

JavaScript and HTML

Make sure to place your JavaScript and HTML in the proper order. In Figure 2-12, the welcoming HTML in ❽ precedes the JavaScript that actually writes the date and time in ❾, since the browser first writes that text and then executes the JavaScript. With JavaScript, as with HTML, browsers read from the top of the page down. I've put `document.write()` in the body so that the actual date information will come after the welcome text. I've put the rest of the JavaScript at the head of the page to keep the body HTML cleaner.

Why document.write()?

Notice that the code in Figure 2-11 uses `document.write()` instead of `window.document.write()`. In general, it's fine to drop the word `window` and the first dot before the word `document`. In future chapters I'll tell you when the word `window` must be added.

How the European Space Agency Writes the Date to Its Page

The JavaScript used by the European Space Agency is very much like the code I used for the *Book of JavaScript* web page. One big difference between the two is that the ESA prints out the month using abbreviations like *Jan* and *Feb* for *January* and *February*. They do this using arrays, a topic discussed in Chapter 8, so in Figure 2-13 I've modified their code a bit to focus on topics covered so far.