

# Strings, Math, and Dates

---

For most of the lessons in the tutorial so far, the objects at the center of attention were objects belonging to the document object model. But as indicated in Chapter 2, a clear dividing line exists between the document object model and the JavaScript language. The language has some of its own objects that are independent of the document object model. These objects are defined such that if a vendor wished to implement JavaScript as the programming language for an entirely different kind of product, the language would still use these core facilities for handling text, advanced math (beyond simple arithmetic), and dates.

## Core Language Objects

It is often difficult for newcomers to programming or even experienced programmers who have not worked in object-oriented worlds before to think about objects, especially when attributed to “things” that don’t seem to have a physical presence. For example, it doesn’t require lengthy study to grasp the notion of a button on a page being an object. It has several physical properties that make perfect sense. But what about a string of characters? As you learn in this chapter, in an object-based environment such as JavaScript, everything that moves is treated as an object — each piece of data from a Boolean value to a date. Each such object probably has one or more properties that help define the content; such an object may also have methods associated with it to define what the object can do or what can be done to the object.

I call all objects not part of the document object model *global objects*. You can see the full complement of them in the JavaScript Object Road Map in Appendix A. In this lesson the focus is on the String, Math, and Date objects.

## String Objects

You have already used string objects many times in earlier lessons. A string is any text inside a quote pair. A quote pair

# 10

C H A P T E R



### In This Chapter

How to modify strings with common string methods

When and how to use the Math object

How to use the Date object



consists of either double quotes or single quotes. This allows one string to be nested inside another, as often happens in event handlers. In the following example, the `alert()` method requires a quoted string as a parameter, but the entire method call must also be inside quotes.

```
onClick="alert('Hello, all!')"
```

JavaScript imposes no practical limit on the number of characters that a string can hold. However, most browsers have a limit of 255 characters in length for a script statement. This limit is sometimes exceeded when a script includes a lengthy string that is to become scripted content in a page. Such lines need to be divided into smaller chunks using techniques described in a moment.

You have two ways to assign a string value to a variable. The simplest is a simple assignment statement:

```
var myString = "Howdy"
```

This works perfectly well except in some exceedingly rare instances. Beginning with Navigator 3 and Internet Explorer 4, you can also create a string object using the more formal syntax that involves the `new` keyword and a constructor function (that is, it “constructs” a new object):

```
var myString = new String("Howdy")
```

Whichever way you use to initialize a variable with a string, the variable is able to respond to all string object methods.

## Joining strings

Bringing two strings together as a single string is called *concatenating* strings, a term I introduce in Chapter 6. String concatenation requires one of two JavaScript operators. Even in your first script in Chapter 3, you saw how the addition operator linked multiple strings together to produce the text dynamically written to the loading Web page:

```
document.write(" of <B>" + navigator.appName + "</B>.")
```

As valuable as that operator is, another operator can be even more scripter-friendly. The situation that calls for this operator is when you are building large strings. The strings are either so long or cumbersome that you need to divide the building process into multiple statements. Some of the pieces may be string literals (strings inside quotes) or variable values. The clumsy way to do it (perfectly doable in JavaScript) is to use the addition operator to append more text to the existing chunk:

```
var msg = "Four score"  
msg = msg + " and seven"  
msg = msg + " years ago,"
```

But another operator, called the add-by-value operator, offers a handy shortcut. The symbol for the operator is a plus and equal sign together. The operator means “append the stuff on the right of me to the end of the stuff on the left of me.” Therefore, the above sequence would be shortened as follows:

```
var msg = "Four score"  
msg += " and seven"  
msg += " years ago,"
```

You can also combine the operators if the need arises:

```
var msg = "Four score"  
msg += " and seven" + " years ago"
```

I use the add-by-value operator a lot when accumulating HTML text to be written to the current document or another window.

## String methods

Of all the JavaScript global objects, the string object has the most diverse collection of methods associated with it. Many methods are designed to help scripts extract segments of a string. Another group, rarely used in my experience, lets methods style text for writing to the page (a scripted equivalent of tags for font size, style, and the like).

To use a string method, the string being acted upon becomes part of the reference, followed by the method name. All methods return a value of some kind. At times, the returned value is a converted version of the string object referred to in the method call. Therefore, if the returned value is not being used directly as a parameter for some method or function call, it is vital that the returned value is caught by a variable:

```
alert(string.methodName())  
var result = string.methodName()
```

The following sections introduce you to several important string methods available to all browser brands and versions.

### Changing string case

A pair of methods convert a string to all uppercase or lowercase letters:

```
var result = string.toUpperCase()  
var result = string.toLowerCase()
```

Not surprisingly, you must observe the case of each letter of the method names if you want them to work. These methods come in handy for times when your scripts need to compare strings that may not have the same case (for example, a string in a lookup table compared to a string typed by a user). Because the methods don't change the original strings attached to the expressions, you can simply compare the evaluated results of the methods:

```
var foundMatch = false  
if (stringA.toUpperCase() == stringB.toUpperCase()) {  
    foundMatch = true  
}
```

### String searches

You can use the `string.indexOf()` method to determine if one string is contained by another. Even within JavaScript's own object data this can be useful

information. For example, another property of the navigator object you used in Chapter 3 (`navigator.userAgent`) reveals a lot about the browser that has loaded the page. A script can investigate the value of that property for the existence of, say, “Win” to determine that the user has a Windows operating system. That short string might be buried somewhere inside a long string, and all the script needs to know is whether the short string is present in the longer one, wherever it might be.

The `string.indexOf()` method returns a number indicating the index value (zero based) of the character in the larger string where the smaller string begins. The key point about this method is that if no match occurs, the returned value is `-1`. To find out whether the smaller string is inside, all you need to test is whether the returned value is something other than `-1`.

Two strings are involved with this method, the shorter one and the longer one. The shorter string is the one that appears in the reference to the left of the method name; the longer string is inserted as a parameter to the `indexOf()` method. To demonstrate the method in action, the following fragment looks to see if the user is running Windows:

```
var isWindows = false
if ("Win".indexOf(navigator.userAgent) != -1) {
    isWindows = true
}
```

The operator in the `if` construction’s condition (`!=`) is the inequality operator. You can read it as meaning “is not equal to.”

### Extracting characters and substrings

To extract a single character at a known position within a string, use the `charAt()` method. The parameter of the method is an index number (zero based) of the character to extract. When I say “extract,” I don’t mean delete, but rather grab a snapshot of the character. The original string is not modified in any way.

For example, consider a script in a main window that is capable of inspecting a variable, `stringA`, in another window that shows maps of different corporate buildings. When the window has a map of Building C in it, the `stringA` variable contains “Building C”. Since the building letter is always at the tenth character position of the string (or number 9 in a zero-based counting world), the script can examine that one character to identify the map currently in that other window:

```
var stringA = "Building C"
var oneChar = stringA.charAt(9)
// result: oneChar = "C"
```

Another method — `string.substring()` — lets you extract a contiguous sequence of characters, provided you know the starting and ending positions of the substring you want to grab a copy of. Importantly, the character at the ending position value is not part of the extraction: All applicable characters up to *but not including* that character are part of the extraction. The string from which the extraction is made appears to the left of the method name in the reference. Two parameters specify the starting and ending index values (zero based) for the start and end positions:

```
var stringA = "banana daiquiri"
```

```
var excerpt = stringA.substring(2,6)
// result: excerpt = "nana"
```

String manipulation in JavaScript is fairly cumbersome compared to some other scripting languages. Higher level notions of words, sentences, or paragraphs are completely absent. Therefore, sometimes it takes a bit of scripting with string methods to accomplish what would seem like a simple goal. And yet you can put your knowledge of expression evaluation to the test as you assemble expressions that utilize heavily nested constructions. For example, the following fragment needs to create a new string that consists of everything from the larger string except the first word. Assuming the first word of other strings could be of any length, the second statement utilizes the `string.indexOf()` method to look for the first space character and adds 1 to that value to serve as the starting index value for an outer `string.substring()` method. For the second parameter, the `length` property of the string provides a basis for the ending character's index value (one more than the actual character needed).

```
var stringA = "The United States of America"
var excerpt = stringA.substring((stringA.indexOf(" ") + 1,
stringA.length)
// result: excerpt = "United States of America"
```

Since creating statements like this one is not something you are likely to enjoy over and over again, I show you in Chapter 26 how to create your own library of string functions you can reuse in all of your scripts that need their string-handling facilities.

## The Math Object

JavaScript provides ample facilities for math — far more than most scripters who don't have a background in computer science and math will use in a lifetime. But every genuine programming language needs these powers to accommodate clever programmers who can make windows fly in circles on the screen.

All of these powers are contained by the Math object. This object is unlike most of the rest in JavaScript in that you don't generate copies of the object to use. Instead your scripts summon a single Math object's properties and methods (inside Navigator, one Math object actually occurs per window or frame, but this has no impact whatsoever on your scripts). That Math object (with an uppercase M) is part of the reference to the property or method. Properties of the Math object are constant values, such as pi and the square root of two:

```
var piValue = Math.PI
var rootOfTwo = Math.SQRT2
```

Methods cover a wide range of trigonometric functions and other math functions that work on numeric values already defined in your script. For example, you can find which of two numbers is the larger:

```
var larger = Math.max(value1, value2)
```

Or you can raise one number to a power of ten:

```
var result = Math.pow(value1, power1)
```

More common, perhaps, is the method that rounds a value to the nearest integer value:

```
var result = Math.round(value1)
```

Another common request of the `Math` object is a random number. Although the feature was broken on Windows and Macintosh versions of Navigator 2, it works in all other versions and brands since. The `Math.random()` method returns a floating-point number between 0 and 1. If you are designing a script to act like a card game, you need random integers between 1 and 52; for dice, the range is 1 to 6 per die. To generate a random integer between zero and any top value, use the following formula:

```
Math.round(Math.random() * n)
```

where `n` is the top number. To generate random numbers between a different range use this formula:

```
Math.round(Math.random() * n) + m
```

where `m` is the lowest possible integer value of the range and `n` equals the top number of the range minus `m`. In other words `n+m` should add up to the highest number of the range you want. For the dice game, the formula for each die would be

```
newDieValue = Math.round(Math.random() * 5) + 1
```

One bit of help JavaScript doesn't offer is a way to specify a number formatting scheme. Floating-point math can display more than a dozen numbers to the right of the decimal. Moreover, results can be influenced by each operating system's platform-specific floating-point errors, especially in earlier versions of scriptable browsers. Any number formatting — for dollars and cents, for example — must be done through your own scripts. An example is provided in Chapter 27.

## The Date Object

Working with dates beyond simple tasks can be difficult business in JavaScript. A lot of the difficulty comes with the fact that dates and times are calculated internally according to Greenwich mean time (GMT) — provided the visitor's own internal PC clock and control panel are set accurately. As a result of this complexity, better left for Chapter 28, this section of the tutorial touches on only the basics of the JavaScript Date object.

A scriptable browser contains one global `Date` object (in truth one `Date` object per window) that is always present, ready to be called upon at any moment. When you wish to work with a date, such as displaying today's date, you need to invoke the `Date` object constructor to obtain an instance of a date object tied to a specific time and date. For example, when you invoke the constructor without any parameters, as in

```
var today = new Date()
```

the `Date` object takes a snapshot of the PC's internal clock and returns a date object for that instant. The variable, `today`, contains not a ticking clock, but a value that can be examined, torn apart, and reassembled as needed for your script.

Internally, the value of a date object instance is the time, in milliseconds, from zero o'clock on January 1, 1970, in the Greenwich mean time zone — the world standard reference point for all time conversions. That's how a date object contains both date and time information.

You can also grab a snapshot of the `Date` object for a particular date and time in the past or future by specifying that information as parameters to the `Date` object constructor function:

```
var someDate = new Date("Month dd, yyyy hh:mm:ss")
var someDate = new Date("Month dd, yyyy")
var someDate = new Date(yy,mm,dd,hh,mm,ss)
var someDate = new Date(yy,mm,dd)
```

If you attempt to view the contents of a raw date object, JavaScript automatically converts the value to the local time zone string as indicated by your PC's control panel setting. To see this in action, use Navigator to go to the `javascript:` URL where you can type in an expression to see its value. Enter the following:

```
new Date()
```

The current date and time as your PC's clock calculates it is displayed (even though JavaScript still stores the date object's millisecond count in the GMT zone). You can, however, extract components of the date object via a series of methods that you can apply to the date object instance. Table 10-1 shows an abbreviated listing of these properties and information about their values.

Be careful about values whose ranges start with zero, especially the months. The `getMonth()` and `setMonth()` method values are zero-based, so the numbers will be one less than the month numbers you are accustomed to working with (for example, January is 0, December is 11).

Table 10-1  
Some Date Object Methods

<i>Method</i>	<i>Value Range</i>	<i>Description</i>
<code>dateObj.getTime()</code>	0-...	Milliseconds since 1/1/70 00:00:00 GMT
<code>dateObj.getYear()</code>	70-...	Specified year minus 1900; 4-digit year for 2000+
<code>dateObj.getMonth()</code>	0-11	Month within the year (January = 0)
<code>dateObj.getDate()</code>	1-31	Date within the month
<code>dateObj.getDay()</code>	0-6	Day of week (Sunday = 0)
<code>dateObj.getHours()</code>	0-23	Hour of the day in 24-hour time

(continued)

Table 10-1 (*continued*)

<i>Method</i>	<i>Value Range</i>	<i>Description</i>
<code>dateObj.getMinutes()</code>	0-59	Minute of the specified hour
<code>dateObj.getSeconds()</code>	0-59	Second within the specified minute
<code>dateObj.setTime(val)</code>	0-...	Milliseconds since 1/1/70 00:00:00 GMT
<code>dateObj.setYear(val)</code>	70-...	Specified year minus 1900; 4-digit year for 2000+
<code>dateObj.setMonth(val)</code>	0-11	Month within the year (January = 0)
<code>dateObj.setDate(val)</code>	1-31	Date within the month
<code>dateObj.setDay(val)</code>	0-6	Day of week (Sunday = 0)
<code>dateObj.setHours(val)</code>	0-23	Hour of the day in 24-hour time
<code>dateObj.setMinutes(val)</code>	0-59	Minute of the specified hour
<code>dateObj.setSeconds(val)</code>	0-59	Second within the specified minute

You may notice one difference about the methods that set values of a date object. Rather than returning some new value, these methods actually modify the value of the date object referenced in the call to the method.

## Date Calculations

Performing calculations with dates requires working with the millisecond values of the date objects. This is the surest way to add, subtract, or compare date values. To demonstrate a few date object machinations, Listing 10-1 displays the current date and time as the page loads. Another script calculates the date and time seven days from the current date and time value.

In the Body portion, the first script runs as the page loads, setting a global variable, `today`, to the current date and time. The string equivalent is written to the page. In the second Body script, the `document.write()` method invokes the `nextWeek()` function to get a value to display. That function utilizes the `today` global variable, copying its millisecond value to a new variable, `todayInMS`. To get a date 7 days from now, the next statement simply adds the number of milliseconds in 7 days (60 seconds times 60 minutes times 24 hours times 7 days times 1000 milliseconds) to today's millisecond value. The script now needs a new date object calculated from the total milliseconds. This requires invoking the Date object constructor with the milliseconds as a parameter. The returned value is a date object, which is automatically converted to a string version for writing to the page. Letting JavaScript create the new date with the accumulated number of milliseconds is more accurate than trying to add 7 to the value returned by the date object's `getDate()` method. JavaScript automatically takes care of figuring out how many days there are in a month and leap years.



**Listing 10-1: Date Object Calculations**

```
<HTML>
<HEAD>
<TITLE>Date Calculation</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function nextWeek() {
    var todayInMS = today.getTime()
    var nextWeekInMS = todayInMS + (60 * 60 * 24 * 7 * 1000)
    return new Date(nextWeekInMS)
}
</SCRIPT>
</HEAD>

<BODY>
Today is:
<SCRIPT LANGUAGE="JavaScript">
var today = new Date()
document.write(today)
</SCRIPT>
<BR>
Next week will be:
<SCRIPT LANGUAGE="JavaScript">
document.write(nextWeek())
</SCRIPT>
</BODY>
</HTML>
```

---

Many other quirks and complicated behavior await you if you script dates in your page. As later chapters demonstrate, however, it may be worth the effort.

## Exercises

1. Create a Web page that has one form field for entry of the user's e-mail address and a Submit button. Include a presubmission validation routine that verifies that the text field has the @ symbol found in all e-mail addresses before allowing the form to be submitted.
2. Given the string "Netscape Navigator," fill in the blanks of the `string.substring()` method parameters here that yield the results shown to the right of each method call:

```
var myString = "Netscape Navigator"
myString.substring(__,__) // result = "Net"
myString.substring(__,__) // result = "gator"
myString.substring(__,__) // result = "cape Nav"
```

- 3. Fill in the rest of the function in the listing that follows that would look through every character of the entry field and count how many times the letter “e” appears in the field. (Hint: All that is missing is a `for` repeat loop.)**

```
<HTML>
<HEAD>
<TITLE>Wheel o' Fortuna</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function countE(form) {
    var count = 0
    var inputString = form.mainstring.value.toUpperCase()
    missing code
    alert("The string has " + count + " instances of the letter e.")
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
Enter any string: <INPUT TYPE="text" NAME="mainstring" SIZE=30><BR>
<INPUT TYPE="button" VALUE="Count the Es" onClick="countE(this.form)">
</BODY>
</HTML>
```

- 4. Create a page that has two fields and one button. The button is to trigger a function that generates two random numbers between 1 and 6, placing each number in one of the fields. (This is used as a substitute for rolling a pair of dice in a board game.)**
- 5. Create a page that displays the number of days between today and next Christmas.**

