chapter **three**

Data Types, Variables, and Literals

Too often we forget that genius . . . depends upon the data within its reach, that Archimedes could not have devised Edison's inventions.

—Ernest Dimnet

Every programming language needs to be able to handle and manipulate data. JavaScript is no different in this respect. Programmers need a means of inputting data, temporarily storing it, modifying it, and outputting it. It helps if data can be categorized by type as well: this is a number, this is a string (text), etc.

In this chapter, we'll

- Look at the various data types and special values that JavaScript recognizes.
- Define the terms "literal" and "variable."
- Specify how to declare and initialize a variable.
- Examine JavaScript's variable-naming rules and establish some acceptable naming conventions.
- Define "expression evaluation."
- See how being loosely typed makes it easy for JavaScript to convert one data type to another.
- Learn how to convert a variable from one data type to another.
- See how to determine a variable's true data type.

- Examine why number, string, and Boolean are primitive data types, built-in functions, and objects.

Let's get started.

JavaScript Data Types

JavaScript supports three **primitive data types**—number, string, and Boolean—and one **composite data type**—Object. By "primitive," I mean they can't get any simpler. By "composite," I mean that it is made up of a combination of data, usually of various types. You know that an object has properties that define and describe it and methods on which it can act. Thus, an object is a composite of the data defining and stored in its properties and the data defining its methods. In addition to these data types, JavaScript recognizes four special values: `null`, `undefined`, `NaN`, and `Infinity`. Let's examine each of these data types and special values in turn.

Number

A **number** is any numeric value, be it a floating-point number (float) such as 4.17 or -32.518 or a whole number (integer) like –55 or 187. For the most part, JavaScript does not distinguish between the two, although it is possible to convert a string specifically to an integer or a float. More about that later.

JavaScript represents numbers using the eight-byte, floating-point, numeric format standard established by the Institute of Electrical and Electronics Engineers, Inc. (IEEE). Say that three times fast! What it means is that you can represent very large numbers up to $\pm 1.7976931348623157 \text{x} 10^{308}$ and very small numbers down to $\pm 5.0 \text{x} 10^{-324}$. "Very large" is a bit of an understatement, we're talking *huge* here; $\pm 1.79 \text{x} 10^{308}$ is bigger than a centillion! To better wrap your mind around the size of this number, visit http://mathworld.wolfram.com/LargeNumber.html for a list of large numbers and their English word equivalents. I couldn't find a word to describe a number as small as $\pm 5 \text{x} 10^{-324}$, but I can tell you that it is very, very tiny indeed. It is highly unlikely you'll need to reference a number smaller than that.

When it comes to working with numbers in JavaScript, you're not stuck working with the decimal (base 10) number system alone. JavaScript also recognizes and supports the hexadecimal (base 16) and the octal (base 8) number systems and lets you enter integers in those formats, as well as the normal decimal format. Because decimal is the most popular and often used number system, to indicate a number in decimal, you simply write the number as you normally would without a leading zero (0). Hexadecimal and octal are a little different. To enter a number in hexadecimal, precede the number with 0x (zero x). For octal numbers, use a leading zero (0). Here's a summary. The examples in the following table all evaluate to 127 decimal.

Number System	Base	Notation	Example
decimal	10	Enter the number as a normal integer without a leading 0 (zero).	127
hexadecimal	16	Enter the number as an integer with a leading 0x (zero x) or 0X (zero X).	0x7F or 0X7F
octal	8	Enter the number as an integer with a leading 0 (zero).	0177

Table 3.1 Specifying Integer Bases in JavaScript

JavaScript is also flexible about how you designate floating-point numbers. You can write floating-point numbers in the usual way, as an integer followed by a decimal point and a fraction expressed in decimal, or you can write them in scientific notation.

Method	Notation	Examples
normal	Enter the number as a decimal integer followed by a decimal point (.) and a fraction expressed in decimal.	12000000 -7.35 .552
scientific notation	Enter the number as a decimal integer followed by an exponent indicator (E) and an integer that can be signed (preceded by a "+" or "-").	12E6 -.735E1 .552E0

Table 3.2 Specifying Floating-Point Numbers in JavaScript

So numbers are simply numbers. That's easy. But what's a string?

String

A **string** is text, that is, any combination or *string* of letters, numbers, punctuation, etc. Maybe that's where the name came from: it's a *string* of characters. However it got its name, a string is easily recognized because it is always contained within or delimited by quotation marks. JavaScript allows you to use either double quotation marks or single quotation marks, also known as apostrophes, to delimit strings. Here are some examples of strings:

```
"WebWoman"
"3.141592"
'Greetings, Earthling'
"onClick=alert('Hello, World')"
'2, 4, 6, 8, who do we appreciate?'
"What\'s up?"
"Tina says, \"She who laughs, lasts!\""
```

Notice the odd backslashes in the last two examples? Why are they there? Because quotation marks, both double and single, are special characters used by the JavaScript language to delimit strings. You have to **escape** them in order to use them as part of a string; that is, you have to preface each with a backslash character in order to *escape* its special meaning to JavaScript. Look at the next to last example above. The apostrophe (single quotation mark) has been escaped. If you were to write that string out in JavaScript using the document's write method,

```
document.write("What\'s up?")
```

the result would be

```
What's up?
```

The same is true for the last example, that is, the quotation marks in the string have been escaped so that the statement

```
document.write("Tina says, \"She who laughs, lasts!\"")
```

results in

```
Tina says, "She who laughs, lasts!"
```

If you didn't escape the apostrophe and double quotation marks, you could get an error when you run the code. For every opening single or double quotation mark, JavaScript expects a corresponding closing quotation mark.

Many HTML attribute values must be enclosed in quotation marks. In fact, any HTML attribute value that contains any character other than a number or a letter must be delimited with quotation marks. Some older browsers do not recognize single quotation marks, so it is a good habit to only use double quotation marks around HTML attribute values. Because you often use JavaScript to write HTML statements, it is important that you know how to escape quotation marks. Here's another example, this time writing HTML:

```
document.write("<a href=\"http://www.webwoman.biz/\">WebWoman</a>")
```

The `href` attribute value in the code above contains a colon (:) and several forward slashes (/) so it *must* be delimited by double quotation marks (an HTML requirement). The result of the above statement is that the following HTML code is written to the document:

```
<a href="http://www.webwoman.biz/">WebWoman</a>
```

OK, so both single and double quotation marks have special meaning to JavaScript. Are there any others you need to worry about? As a matter of fact, there are.

JavaScript Special Characters

The backslash character (\) itself has a special meaning to JavaScript. It escapes a character from its special JavaScript meaning. The backslash, single quotation mark, and double quotation mark are the only characters you need to worry about escaping. However, there's actually a whole list of escape sequences you can use in JavaScript. I've listed the most often used sequences first.

Escape Sequence	Character Represented
`\"`	Double quotation mark (“)
`\'`	Single quotation mark or apostrophe (‘)
`\\`	Backslash (\\)
`\n`	New line—causes following text to begin on a new line. Particularly useful in alert message strings.
`\x`*HH*	The character with the Latin-1 encoding specified by two hexadecimal digits, *HH*. The hexadecimal number must fall between 00 and FF, inclusive. For example, \xA9 is the hexadecimal sequence for the copyright symbol.
`\u`*XXXX*	The Unicode character specified by four hexadecimal digits, *XXXX*. For example, \u00A9 is the Unicode sequence for the copyright symbol.
`\`*XXX*	The character with the Latin-1 encoding specified by up to three octal digits, *XXX* . The octal number must fall between 0 and 377, inclusive. For instance, \251 is the octal sequence for the copyright symbol.
`\b`	Backspace
`\f`	Form feed
`\r`	Carriage return
`\t`	Tab

Table 3.3 JavaScript Escape Sequences and Special Characters

In addition to numbers and strings, JavaScript supports Boolean data values.

Boolean (Logical)

A **Boolean** value is a truth value that specifies whether something is true or false. While there are an infinite number of possible values for the string and number data types, there are only two possible values for the Boolean data type: true and false. Notice the absence of quotation marks around true and false. Some languages, like C, do not have a specific Boolean or logical data type, as it is often called. Instead, they often use the integers 1 (true) and 0 (false) to represent truth values. JavaScript, however, does recognize Boolean as a distinct data type. You'll see how useful this data type is when you begin working with comparison and logical operators in Chapter 4 and when you learn about control structures in Chapter 6.

Object

An **Object** is a composite data type, that is, it does not hold just one primitive type of data value, such as number, string, or Boolean. Instead it is *composed* of zero or more pieces of data, each of which may be of a different basic data type. For example, an image's `src` property is a string, whereas its `width` property is a number and its `complete` property is a Boolean. We'll continue to discuss and use objects throughout this text. Chapter 16 provides details for creating your own custom objects. For now, just file it away as another data type.

Special Values

In addition to the data types discussed above, JavaScript recognizes four special values: `null`, `undefined`, `Infinity`, and `NaN`.

- `null` is a special keyword that represents no value, nothing.
- `undefined` is a special keyword indicating that the value has not been defined. You'll run across this nasty value whenever you try to write out or work with a variable that has been declared, but not initialized, and has never had a value assigned to it. We'll define the terms *variable, declare,* and *initialize* in the next section.
- `Infinity` is a numeric value representing infinity. You'll usually encounter it when you perform mathematical computations that result in an infinite or undefined value. For instance, divide any number by zero and you'll get `Infinity`. However, in really old browsers, you may get `undefined` or `NaN` instead.
- `NaN` is a special value indicating that the result is "not a number." We'll look more closely at it a bit later in this chapter.

To summarize, all data handled by JavaScript will be one of the above-listed primitive data types—number, string, or Boolean—or one of the above-listed special values—`null`, `undefined`, `Infinity`, or `NaN`. Every piece of data that you can work with, whether it be a variable, a literal, an object property, or the result of an expression evaluation, is, in its most basic sense, one of the following:

- a number
- a string
- a Boolean
- `null` (that is, it has no value)
- `undefined` (that is, it has not yet been defined itself or its value has not yet been assigned)
- `Infinity`
- `NaN`

Literal vs. Variable

What do these terms "literal" and "variable" mean? A **literal** is any value that can be expressed verbatim or *literally*, that is, a literal is a fixed value taken in its basic sense, exactly as written. For instance, the number 10 is a numeric literal, whereas the word "Hello" is a string literal. You use literals throughout your programs in calculations, to print words and phrases on the screen, and much more.

While a literal is a fixed value, a **variable** is a symbolic name that represents a value that can, and likely will, change or *vary* during a program's execution. Physically, it is a storage place in memory that you can access using a symbolic name that you choose when you define or *declare* that variable. When you **declare** a variable, the JavaScript interpreter tells the computer to set aside space in memory to hold the value of your variable.

You may think of a variable as a cup. Let's label the cup `myCup`. Throughout its life, `myCup` may contain a variety of liquids. For instance, in the summer you may fill `myCup` with lemonade, ice water, iced tea, or soda, and in the winter with hot tea, cocoa, or coffee. The contents of `myCup` *vary* according to your needs. Similarly, the contents of a variable can vary according to the needs of your program. Whereas a literal has only one literal, fixed value, a variable's value varies.

Declaring Variables

In order to use a variable, you should first declare it and, preferably, **initialize** it; that is, you should define its name and, optionally, assign it a starting value. Notice that I said "should." While it is good programming practice to declare and initialize a variable before using it in your program, JavaScript does not require you to do so. By initializing it when you declare it, you avoid the risk of errors that can result from attempting to evaluate an undefined value.

To declare a variable, you simply type the special keyword `var`, followed by a space and the name of your variable:

```
var myVariable
```

The above variable declaration tells the computer to allocate space for `myVariable`. It does not specify what type of data you intend to store in `myVariable`, nor does it place a value in the allocated space. It simply sets aside the space and labels it. At this point, the value of `myVariable` is `undefined`.

Here are three more examples:

```
var num1
var firstName
var isOn
```

You can also declare two or more variables at once in a single JavaScript statement: simply separate them with commas. The above example could also be written

```
var num1, firstName, isOn
```

Initializing Variables

To **initialize** a variable is to assign it a default or starting value. The best time to initialize a variable is when you declare it. So how do you do that? Simple, you follow your declaration with an **assignment statement**. An assignment statement is an operation that *assigns* a value to a variable. The most often used assignment operator is = (equals). We'll look at more assignment operators in Chapter 4. Here are some variable initialization examples:

```
var num1 = 37
var firstName = "Tina"
var isOn = false
```

To declare and initialize two or more variables in one fell swoop, simply separate them with commas. The above example could also be written as

```
var num1 = 37, firstName = "Tina", isOn = false
```

You can also initialize a variable with the result of a method or function call:

```
var visitor = prompt("What\'s your name?", "")
```

The above example first prompts the user for her name, then assigns the result of the window prompt method to the variable `visitor`.

Not only is it considered good programming practice to initialize your variables when you declare them, it is also good form to place all of your variable declarations at the beginning of your program and to provide a short description for each variable in a comment. For example:

```
var num1 = 37                  // first number entered
var firstName = "Tina"         // user's first name
var isOn                       // indicates whether Java is
                               //   enabled or not
```

Comments not only make your code more readable for other people, they also help you quickly figure out what you did when you revisit your code six months or a year later. Believe me, if you don't insert comments when you write the code in the first place, you'll be cussing to yourself later when you have to decipher and modify your code.

Current versions of JavaScript do not *require* you to declare a variable before using it, that is, you can introduce and use variables on the fly throughout your program. For instance, in the middle of your program you could start using a variable called `num5`, even though you never declared it:

```
num5 = 99
document.write(num5)
```

JavaScript will neither grumble nor complain. It will simply create your new variable, `num5`, and assign it the value 99. From that point on, you can use that variable in your program. You *can* do it. However, I do *not* recommend that you do it. Declaring your variables before you use them not only shows forethought and helps you plan out your program, but it also makes your code more readable and more easily modifiable. It's much easier and quicker to change variable default values when they're all in one place: at the beginning of your program. Also, when debugging, you won't have to hunt through a ton of code just to discover what values your variables began with.

One more important point to consider: while current versions of JavaScript do not require you to use the keyword `var` to declare a variable, future versions of JavaScript may. So, have I convinced you to follow the conventions I've suggested yet? I hope so.

Variable-Naming Rules

When it comes to *naming* variables, JavaScript has a few rules that you need to keep in mind:

- Variable names may not contain any punctuation, except the underscore (_). Only letters, numbers, and the underscore are valid characters in a variable name.
- Variable names may not begin with a number. They must start with a letter or an underscore (_).
- Variable names may not contain spaces. Thus if you want to use more than one word as a variable name, then you need to mix the case (lastName), separate the words with an underscore (last_name), or simply string them together eliminating the space (lastname).
- Reserved words may not be used as variable names. Reserved words are special words that a programming language sets aside or *reserves* for its own use. Appendix C contains a list of words JavaScript has reserved for its current and future use.
- Pre-defined object, property, method, and built-in function names are also off limits, for example, `document`, `window`, `form`, `name`, `src`, `isNaN`, etc. Consider them reserved words. While you may, technically, sometimes be able to implement them successfully, using pre-defined object, property, method, and function names as variable identifiers is asking for trouble. Not only can it create confusion for anyone trying to read your code, it can also cause error messages, generate undesired side effects or unintended results, or have other unforeseen consequences.

programmer's tip

Personally, I prefer the first option, mixing cases. I find the underscore character difficult to type correctly: I end up with a dash as often as not.The mixed case is also more readable than all lower case.While mixed case is my personal preference, you should choose a convention that works best for you (within the rules, of course) and stick to it.

Rule	Invalid Variable Names	Valid Variable Names
No punctuation except underscore (_)	first-name last-name!	first_name lastName
May not begin with a number	1stName 2be 3rdNumber	firstName _2be num3
No spaces allowed	last name	lastName

No reserved words	case
	class
	package
		myCase
		class2
		zPackage

No object, property, method, or built-in function names	document form	theDocument zForm

Table 3.4 JavaScript Variable-Naming Rules

JavaScript Is Case Sensitive

One last note about naming JavaScript variables: JavaScript is case sensitive. Thus, the names `sitevisitor`, `SITEVISITOR`, `siteVisitor`, `SiteVisitor`, `Sitevisitor`, and `sItevisitor` refer to six *different* variables. Remember this. It's easy to overlook, especially since you're used to HTML being case *insensitive*. For instance, say your program needs to refer to a variable declared as follows:

```
var myVariable = "Greetings!"
```

The correct dot notation to access the variable is

```
myVariable
```

The value of

```
MyVariable
```

on the other hand, is `undefined`. In the above example, there is no such variable named `MyVariable`; we named it `myVariable`, and JavaScript is case sensitive enough to know the difference even if HTML is not.

Evaluating Expressions: Just What Are You *Really* Saying?

Expression evaluation is closely related to data values and variables. To determine the value of a variable, you have to assess, calculate, or *evaluate* the expression that declared it or was later assigned to it. For instance, when you declare and initialize the variable num1

```
var num1 = 37
```

`num1` evaluates to 37. In JavaScript, a variable always evaluates to its value. The expression on the right side of the equal sign, 37, assesses or evaluates to 37. The expression

```
30 + 7
```

also evaluates to 37.

We're used to making such evaluations without thinking too much about it. In our everyday language, we often use and evaluate expressions. For instance, we evaluate the expressions "kitty," "pussy cat," "kitty cat," "kit kat," and "puddy cat" to mean "cat." They're all *expressions* for cat; they all mean or *evaluate to* cat.

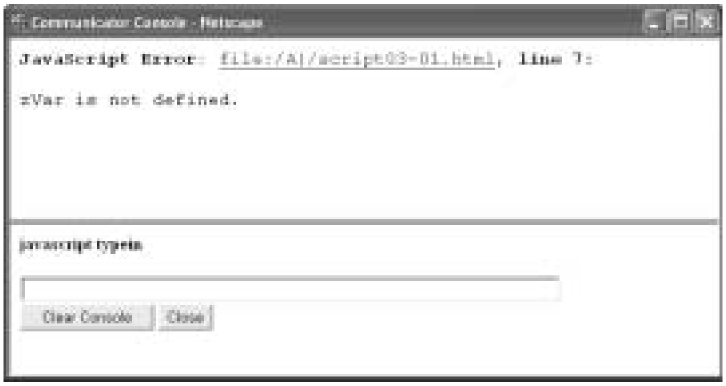
In the next chapter, we'll begin working a lot with expressions as we define the various types of operators that JavaScript supports.

When a variable has not been assigned a value, it is considered **unassigned** and has the value `undefined`. However, when that variable is *evaluated*, the results of the evaluation vary depending on how the variable was declared, or not declared, as the case may be. If the unassigned variable was never really declared formally using the `var` keyword, evaluating it results in a runtime error. For instance, consider Script 3.1:

```
<html>
<head>
      <title>Script 3.1: Using an Undefined Variable</title>
</head>
<body>
<script language="JavaScript" type="text/javascript"><!--
      document.write(zVar)
//-->
</script>
</body>
</html>
```

Script 3.1 Using an Undefined Variable

The variable `zVar` is never formally declared. Although its value is undefined because it is never declared nor initialized, using it results in a runtime error.



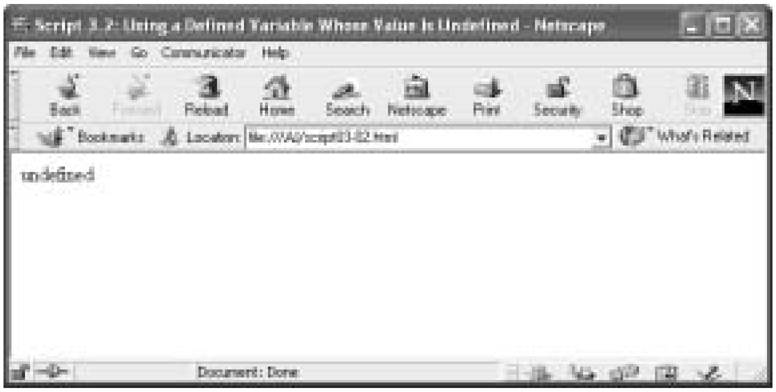
```
JavaScript Error: file:/A|/script03-01.html, line 7:

zVar is not defined.
```

Figure 3.1 Results of Script 3.1—Using an Undefined Variable

On the other hand, if an unassigned variable were formally declared using the `var` keyword, an evaluation of that variable would result in undefined or `NaN`. `NaN` means "not a number." Its use is explained in more detail in a later section of this chapter.

```
<html>
<head>
      <title>Script 3.2: Using a Defined Variable Whose
             Value Is Undefined</title>
</head>
<body>
<script language="JavaScript" type="text/javascript"><!--
      var zVar
      document.write(zVar)
//-->
</script>
</body>
</html>
```

Script 3.2 Using a Defined Variable Whose Value Is Undefined


Script 3.2: Using a Defined Variable Whose Value Is Undefined - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: file:///A|/script03-02.html What's Related
Document: Done

```
undefined
```

Figure 3.2 Results of Script 3.2—Using a Defined Variable Whose Value Is Undefined

Runtime errors certainly are an unwanted side effect. They are a good argument for always formally declaring variables. However, an evaluation of undefined can be just as devastating to a program. For that reason, among others, it is considered good programming practice to initialize all variables when you declare them.

Data Type Conversions: From Numbers to Strings and Back Again

JavaScript is a **loosely typed** language. That means that you do not have to specify the data type of a variable when you declare it or before you use it. In fact, you can actually initialize a variable as a number, then later assign it a string value and still later a Boolean value. For example, the following code is perfectly legal:

```
var guess = 37            // number
guess = "white"           // string
guess = true             // boolean
guess = "pink"            // string
guess = 22                // number
```

Because of this feature, Netscape describes JavaScript as a "**dynamically typed** language." Being dynamically typed has its advantages. For instance, as the above example shows, it is easy to change the data type of a variable on the fly. Simply assign a value of a different data type to the variable.

It is also easy to convert from one data type to another. When a JavaScript expression, involving the plus or concatenation operator, contains numbers followed by strings, JavaScript automatically converts the numbers to string values before evaluating the expression. For example:

```
7 + "up"       // result: 7up
"hi" + 5       // result hi5
```

This is good to know. That means converting a number to a string is easy: just add an empty string to it. For example:

```
var num1 = 85          // initialize num1 as a numeric: 85
num1 = num1 + ""       // now num1 is a string: "85"
```

OK, so converting from a number to a string is easy: just add an empty string to the number. But what if you want to go the other direction: convert from a string to a number? Fortunately, JavaScript has that covered by two very useful built-in functions: `parseInt()` and `parseFloat()`.

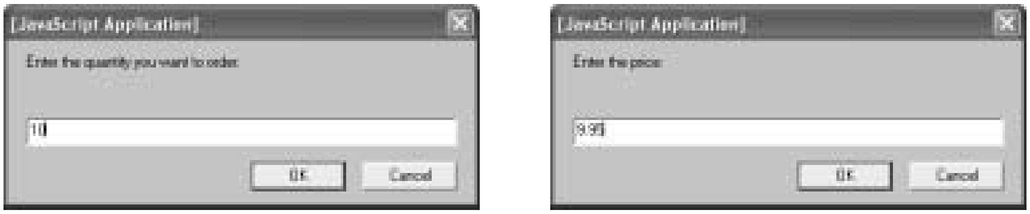
`parseInt()` and `parseFloat()`

When working with Web pages, programmers often need to convert string values to numbers. For instance, did you ever notice that HTML does not provide a *number* box form element? It's true; HTML only provides a *text* box. Similarly, the `window.prompt()` method, which we discussed in Chapter 2, returns a string. So what if you want to perform calculations on data entered in a prompt or text box? Type in the following code:

```
<html>
<head>
      <title>Script 3.3: Prompts Return Strings</title>
</head>
<body>
<script language="JavaScript" type="text/javascript"><!--
      var qty = prompt("Enter the quantity you want to
          order:", "")
      var price = prompt("Enter the price:", "")
      alert("Your total is: " + qty * price)
// -->
</script>
</body>
</html>
```

Script 3.3 Prompts Return Strings

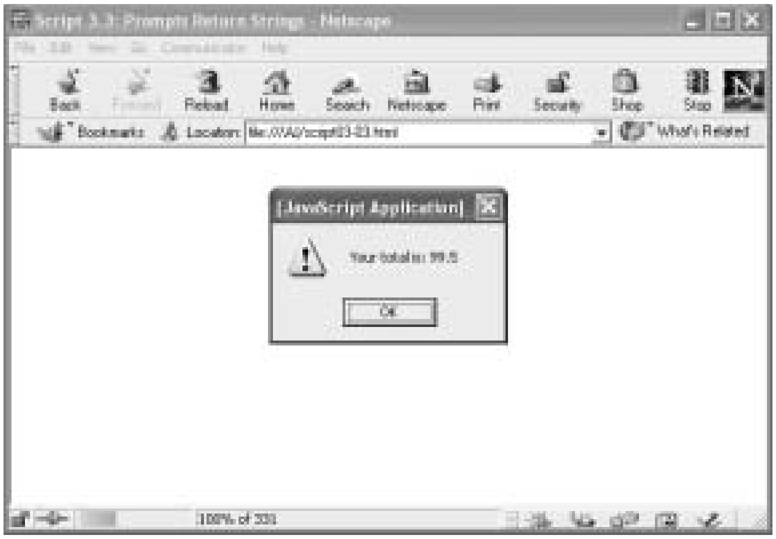
Open the document in your browser and provide a value for quantity and price when prompted:



```
[JavaScript Application]
Enter the quantity you want to order:
10
OK    Cancel

[JavaScript Application]
Enter the price:
9.95
OK    Cancel
```

Figure 3.3 Prompts for Script 3.3

Assuming you didn't make any errors typing in the code, you'll see that, in this case, JavaScript took care of converting the strings you entered in the prompt boxes into numbers before multiplying them.



Your total is: 99.5

Figure 3.4 Final Results of Script 3.3

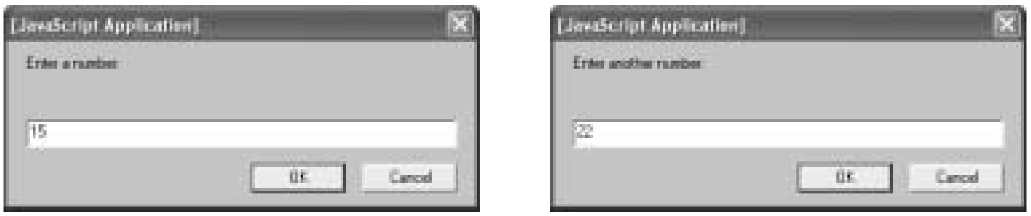
JavaScript recognized that the multiplication operator doesn't apply to strings, so it performed the conversion for you in order to make the expression make sense, in *this* case.

Let's try another example:

```
<html>
<head>
      <title>Script 3.4: Prompts Return Strings</title>
</head>
<body>
<script language="JavaScript" type="text/javascript"><!--
      var num1 = prompt("Enter a number:", "")
      var num2 = prompt("Enter another number:", "")
      alert("Sum: " + (num1 + num2))
// -->
</script>
</body>
</html>
```

Script 3.4 Prompts Return Strings, Version 2

Run the script and enter some numeric values.



Enter a number:

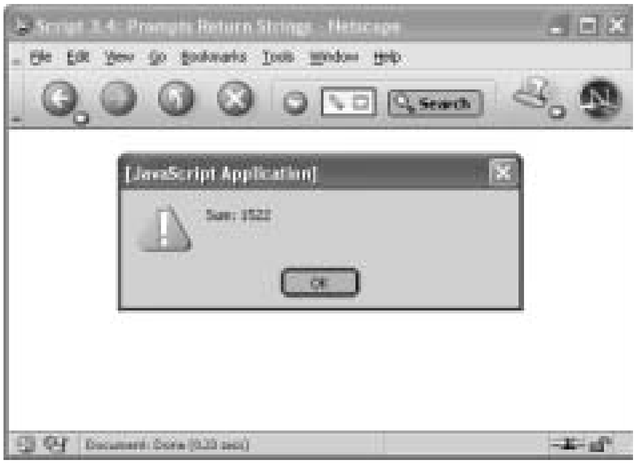
15

Enter another number:

22

Figure 3.5 Prompts for Script 3.4

What happened?


Script 3.4: Prompts Return Strings - Netscape
File Edit View Go Bookmarks Tools Window Help
Search
[JavaScript Application]
OK
Document: Done (0.23 secs)

Sum: 1522

Figure 3.6 Final Results of Script 3.4

This time JavaScript treated both numbers as strings! Why? Because you used the + operator, a legitimate operator for strings, JavaScript did not convert the strings to numbers. Instead, it concatenated them.

When you used the multiplication (*) operator, JavaScript assumed you wanted to convert the strings to numbers and took care of it for you. After all, you can't multiply strings. However, when you used the + operator, which means addition for numbers and concatenation for strings, JavaScript had no way to tell what you wanted to do. So it treated both strings as the *strings* they were and concatenated them. Remember, you entered the numbers into a prompt *text* box.

So how do you get around that? How do you force JavaScript to treat the text box entries as numbers? The answer is that you deliberately convert them

programmer's tip

You *cannot* use `parseInt` to turn a floating-point number into an integer. Neither can you use `parseFloat` to turn an integer into a floating-point number. `parseInt` and `parseFloat` only work on *strings*. Their purpose is to convert strings to numbers.

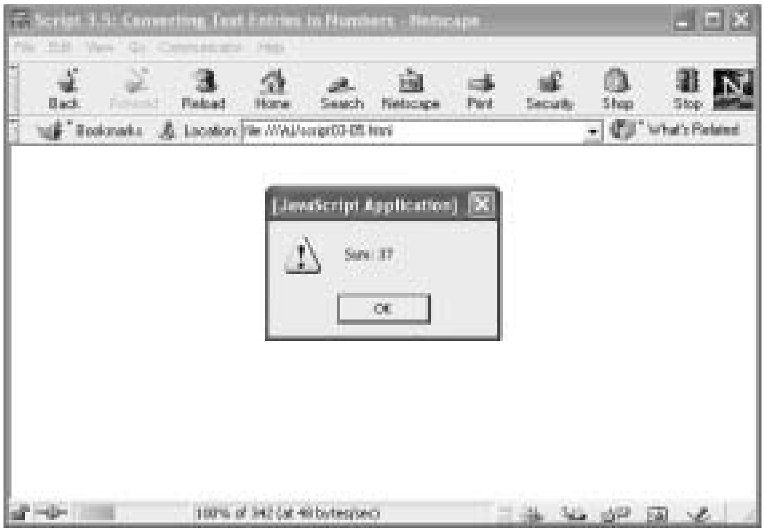
to numbers using `parseInt()` or `parseFloat()` before performing a mathematical operation.

Make the following changes to your script and run it, entering the same numbers as before.

```
<html>
<head>
   <title>Script 3.5: Converting Text Entries to Numbers</title>
</head>
<body>
<script language="JavaScript" type="text/javascript"><!--
   var num1 = prompt("Enter a number:", "")
   var num2 = prompt("Enter another number:", "")
   alert("Sum: " + (parseFloat(num1) + parseFloat(num2)))
// -->
</script>
</body>
</html>
```

Script 3.5 Converting Text Entries to Numbers

Now the program works as originally intended. It adds the entries together instead of concatenating them.


Script 3.5: Converting Text Entries to Numbers - Netscape
File Edit View Go Communicator Help
Back
Forward
Reload
Home
Search
Netscape
Print
Security
Shop
Stop
Bookmarks
Location: file:///A|/script03-05.html
What's Related
[JavaScript Application]
OK
100% of 342 (at 48 bytes/sec)

Sum: 37

Figure 3.7 Results of Script 3.5

You might also prompt the user for a number and convert it on the spot:

```
var num1 = parseFloat(prompt("Enter a number:", """))
```

In the above statement, the visitor is first prompted for a number. Remember, prompt receives the entry as a string data type. `parseFloat` then converts the string into a number. The result is then assigned to `num1`.

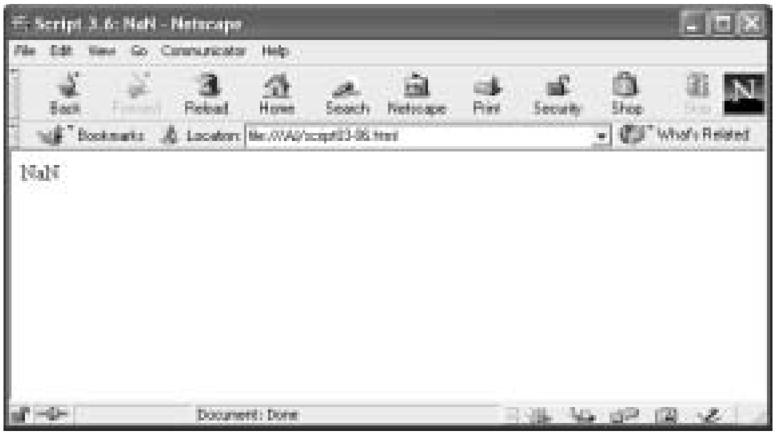
`NaN` and `isNaN()`

What happens if you try to convert a string that does not have a numeric equivalent into a number? For instance, what value would `num1` be set to in the previous example if the visitor entered the string "ten"? Let's look at another example and examine the results:

```
<html>
<head>
      <title>Script 3.6: NaN</title>
</head>
<body>
<script language="JavaScript" type="text/javascript"><!--
      var guess = "white"
      document.write(parseFloat(guess))
// -->
</script>
</body>
</html>
```

Script 3.6 NaN

The result:


Script 3.6: NaN - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: file:///A|/script03-06.html
What's Related
Document: Done

```
NaN
```

Figure 3.8 Results of Script 3.6

NaN is a special keyword that indicates the value is *not a number*. `parseFloat` tried to convert guess into a number, but guess held the value "white," which is a string with no numeric equivalent, so `parseFloat` returned NaN to indicate that it was not a number.

Since `NaN` is a special value, can you use it like the special values `undefined` and `null` in comparisons? Let's try this bit of code and see:

side note

Script 3.7 makes use of an `if` statement. We'll cover the `if` control structure thoroughly in Chapter 6. I've included it here only to illustrate the limitations of `NaN`. For most students, the `if` statement is pretty self-explanatory. After all, you're always making decisions based on some if condition. For instance, "If today is Sunday, do the laundry." Should this code confuse you, please feel free to refer back to it after you've read all about control structures in Chapter 6.

```
<html>
<head>
      <title>Script 3.7: Evaluating
             NaN</title>
</head>
<body>
<script  language="JavaScript"
type="text/javascript"><!--
      var guess = "white"
      guess = parseFloat(guess)
      document.write("Guess: ",
          guess, "<br>")
      if (guess == NaN)    // is guess equal to NaN?
            document.write("It is not a number", "<br>")
// -->
</script>
</body>
</html>
```

Script 3.7 Evaluating NaN

The result:

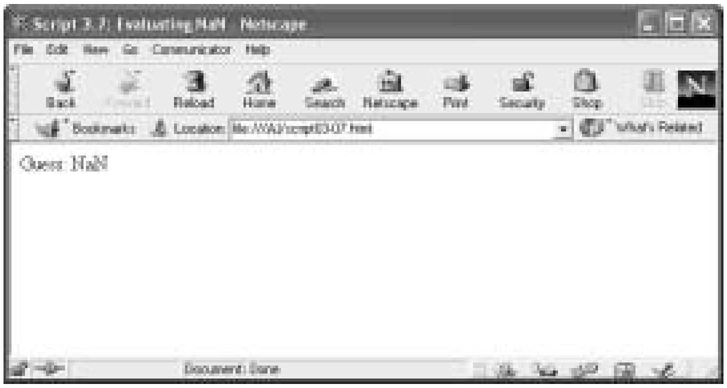

Guess: NaN

Figure 3.9 Results of Script 3.7

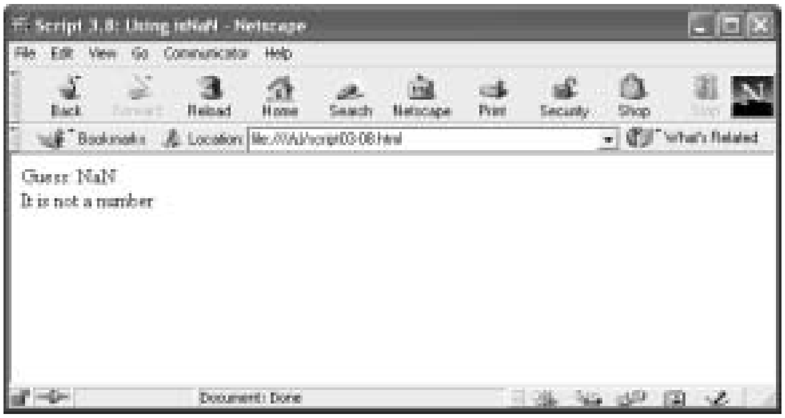
But wait a minute, guess *is* not a number! So why didn't the program write "It is not a number" when it checked to see if guess was equal to `NaN (guess == NaN)`?

Unlike `null` and `undefined`, the special keyword `NaN` cannot be used directly for comparisons. JavaScript does, however, provide a way to see if an expression evaluates to `NaN`. The built-in function to perform this check is—you guessed it—`isNaN()`. With this information in mind, let's modify the script and try it again:

```
<html>
<head>
      <title>Script 3.8: Using isNaN</title>
</head>
<body>
<script language="JavaScript" type="text/javascript"><!--
      var guess = "white"
      guess = parseFloat(guess)
      document.write(guess, "<br>")
      if (isNaN(guess))      // is guess not a number?
            document.write("It is not a number", "<br>")
// -->
</script>
</body>
</html>
```

Script 3.8 Using `isNaN`

This time it worked as we intended:



```
Guess: NaN
It is not a number
```

Figure 3.10 Results of Script 3.8

As you work with JavaScript, and especially with forms, you'll find these built-in functions extremely useful.

Other Built-in Functions for Data Type Conversions

Two other built-in functions, `Number()` and `String()`, can also assist with data type conversions, as can the `toString()` method that is associated with every object.

The `Number` function is not associated with any particular object. It is a built-in function like `parseInt()` and `parseFloat()`. `Number()` attempts to convert any object or string passed to it into a number. If the object sent to it cannot be converted into a number, the function returns `NaN`.

Like the `Number` function, the `String` function is not associated with any particular object. `String()` converts the value of any object into a string. It returns the same value that the object's `toString()` method would.

What Data Type Am I? The `typeof` Operator

One more JavaScript feature that needs mentioning while we're discussing data types is the `typeof` operator. The `typeof` operator lets you easily determine the current data type of any variable. The result of a `typeof` operation is one of the following strings. It indicates the data type of the variable.

- "number"
- "string"
- "boolean"
- "object"
- "undefined"

The syntax is

```
typeof (operand)
typeof operand
```

`operand` is the variable, object, object property, string, or special keyword (null, undefined, etc.) whose type you wish to know.

Note that the parentheses are optional, but it is considered good programming style to use them.

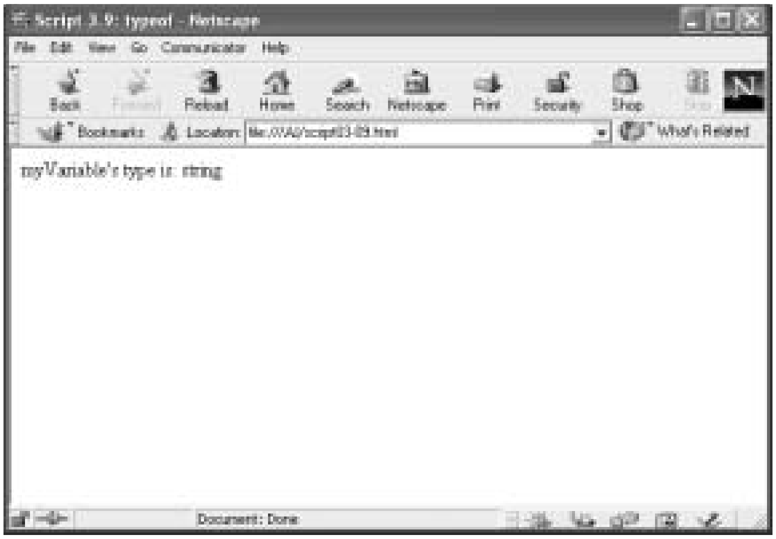
For example:

```
<html>
<head>
      <title>Script 3.9: typeof</title>
</head>
<body>
<script language="JavaScript" type="text/javascript"><!--
      var myVariable = "cat"
      document.write("myVariable\'s type is: ")
      document.write(typeof myVariable, "<br>")
```

```
10  // -->
11  </script>
12  </body>
13  </html>
```

Script 3.9 `typeof`

The result:


Script 3.9: typeof - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: file:///A|/script03-09.html
What's Related
Document: Done

```
myVariable's type is: string
```

Figure 3.11 Results of Script 3.9

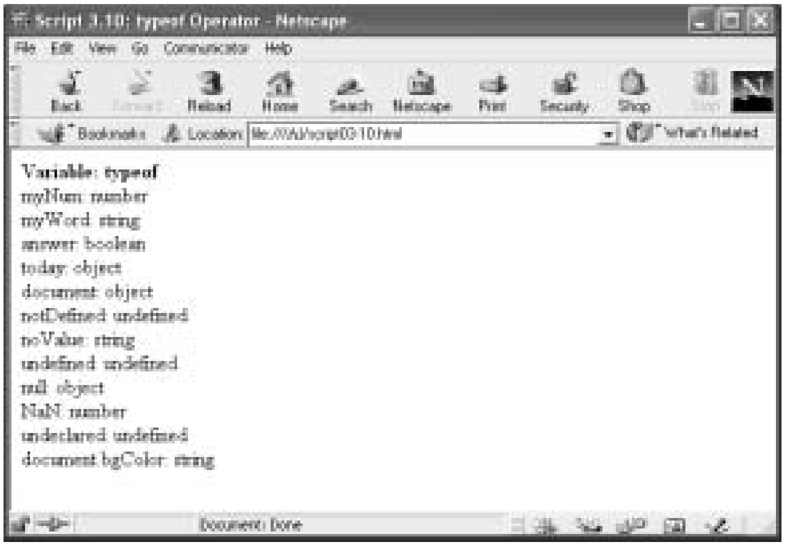
Here's one that's a little more complicated and shows you more of the possible values returned by the `typeof` operator. You're going to have to take line 10, which creates a `Date` object, on faith for now and learn the details later in Chapter 10.

```
1   <html>
2   <head>
3         <title>Script 3.10: typeof Operator</title>
4   </head>
5   <body>
6   <script language="JavaScript" type="text/javascript"><!--
7         var myNum = 7
8         var myWord = "Oh, my!"
9         var answer = false
10        var today = new Date()
11        var notDefined
12        var noValue = ""
13
```

```
        document.write("<b>Variable: typeof</b><br>")
        document.write("myNum: ", typeof(myNum), "<br>")
        document.write("myWord: ", typeof(myWord), "<br>")
        document.write("answer: ", typeof(answer), "<br>")
        document.write("today: ", typeof(today), "<br>")
        document.write("document: ", typeof(document), "<br>")
        document.write("notDefined: ", typeof(notDefined), "<br>")
        document.write("noValue: ", typeof(noValue), "<br>")

        document.write("undefined: ", typeof(undefined), "<br>")
        document.write("null: ", typeof(null), "<br>")
        document.write("NaN: ", typeof(NaN), "<br>")

        document.write("undeclared: ", typeof(undeclared), "<br>")

        document.write("document.bgColor: ",
            typeof(document.bgColor), <br>")
// -->
</script>
</body>
</html>
```

Script 3.10 ***`typeof` Operator***

Here are the results:


Script 3.10: typeof Operator - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: file:///A|/script03-10.html What's Related
Document: Done

```
Variable: typeof
myNum: number
myWord: string
answer: boolean
today: object
document: object
notDefined: undefined
noValue: string
undefined: undefined
null: object
NaN: number
undeclared: undefined
document.bgColor: string
```

Figure 3.12 ***Results of Script 3.10***

Notice that when a `typeof` operation is performed on a variable that has not yet had a value assigned to it, the operation results in "undefined" (see lines 11 and 20 in Script 3.10). "Undefined" is also the result of performing a `typeof` operation on a variable that has not even been declared (line 27). Beware of performing a `typeof` operation on an object that does not exist: the result is a nasty runtime error.

A `typeof` operation on a variable whose value is null or on the keyword null results in "object" (line 24). It seems like a strange response to me.

Interestingly, a value of NaN causes `typeof` to return "number" (line 25). Perhaps the language realizes that you tried to convert the value to a number, and even though it wasn't possible to convert that value to a number, recognizes that you at least *intended* the value to be a number. At least that's my theory.

Clearly, JavaScript makes data type conversions quite easy. Keep in mind, though, that this can also present some potential pitfalls. Here are a few tips that might help you avoid tripping over JavaScript's dynamic data typing:

1. Declare all variables before using them with the keyword `var`, preferably at the beginning of your program.
2. Use descriptive names when defining your variables.
3. Initialize all variables and try to stick to the initial data type throughout your program. Simply initializing all of your variables will help prevent that nasty `undefined` value from popping up in your programs.
4. Describe each variable with a brief comment when you declare and initialize it.
5. When in doubt about whether JavaScript will convert a string to a number for you or not, perform the conversion yourself. That way you'll know for certain that it is taken care of, and you will be less likely to encounter side effects when your script is run in different browsers.
6. Always convert form field entries and prompt box responses for which you expect numeric input into numbers with `parseInt()` or `parseFloat()` and test the result of these operations with `isNaN()` before performing any mathematical operations. You could also use `Number()` to perform the conversion, but it has not been supported as long as `parseInt()` and `parseFloat()` have and will not work in older browsers.

It's an Object, It's a Function, and It's a Primitive Data Type. Huh?

Take a look at your JavaScript object reference in Appendix A. Notice how Number, Boolean, and String are listed as *objects*? But didn't we already say they were *primitive data types*? Didn't we also point out that there are `Number()` and `String()` *functions* as well? So which are they? Primitive data types? Functions? Or objects? The answer is all three.

Number, string, and Boolean are indeed primitive data types. Every value of every property, literal, and variable will boil down to one of these data types or the special values `null` or `undefined`. `Number()` and `String()` are also *functions* that allow you to convert a string to a number and vice versa. Notice that these function names are capitalized and followed by parentheses, unlike most built-in JavaScript function and method names.

Finally, Number, String, and Boolean are pseudo-objects. According to Netscape's JavaScript reference, they are **object wrappers**. What this means is that, for the purposes of using methods associated with these pseudo-objects, a variable will temporarily become the appropriate object for the duration of a method call, then the "object" will be discarded from memory. In other words, the object *wraps* around the primitive data type, temporarily making it an object so that you can use a method defined for that object on the primitive data value. After you're done using the method, the object wrapper is stripped off and discarded like a wet suit.

For example, the String object, in particular, has many useful methods defined for it that allow you to manipulate, tear apart, put back together, uppercase, lowercase, strike out, and even search strings. JavaScript allows you to use all of these String object methods on ordinary strings. For example:

```
var response = "blue"
response = response.toUpperCase()    // result "BLUE"
```

JavaScript automatically converts the string to a temporary String object. Then, after calling the toUpperCase() method, it discards the temporary String object from memory.

Another useful side effect of JavaScript's treating ordinary strings as objects is that you can use the `length` property of the String object to determine the length of any string. For example:

```
response.length             // result 4
```

Keep in mind, however, that ordinary strings are not really objects. We can verify that this is true by using the typeof operator on the string we created:

```
typeof(response)            // result string
```

If for some reason you need a genuine String Object, then you can create one using the `new` operator:

```
var genuineString = new String("The real McCoy")
```

The same holds true for creating genuine number and Boolean objects. We'll discuss the new operator in detail in Chapter 10 and string methods in Chapter 9. You can refer back to this section then for a better understanding of what I just said.

Summary

JavaScript, like any other programming language, has a predefined set of data types and special values that it is able to recognize and manipulate. JavaScript's primitive data types include number, string, and Boolean. The two special data values JavaScript recognizes are `null`, representing no value, and `undefined`, representing the value of a variable that has been declared, but not yet initialized, or the value of a variable or property that simply has not been defined. JavaScript also supports one composite data type: object.

JavaScript is a dynamically typed language. A programmer may easily change the data type of a variable. Strongly typed languages like Java and C++ do not allow this. JavaScript further facilitates dynamic typing by providing several built-in functions for converting data types, including `parseInt()`, `parseFloat()`, `String()`, and `Number()`. In addition, every object has a built-in `toString()` method for converting the object into a string. The `typeof` operator lets you determine the data type of any variable, and the `isNaN()` built-in function lets you verify that a string converted to a number did indeed result in a number.

Finally, JavaScript defines Number, String, and Boolean objects that are really object wrappers that allow you to treat variables with primitive data values as objects for the purpose of using the methods and properties associated with those objects.

Review Questions

1. What does it mean to be a primitive data type?
2. What is a composite data type?
3. What three primitive data types does JavaScript support?
4. What composite data type does JavaScript support?
5. What special values does JavaScript support? List and describe each one.
6. What number bases can JavaScript recognize?
7. How do you differentiate between number bases in your code?
8. What is a string?
9. How does JavaScript recognize strings as strings?
10. What symbols are used to delimit strings in JavaScript?
11. What symbol does HTML prefer to delimit attribute values?
12. What does it mean to "escape a character"? Provide an example.
13. What's the difference between a literal and a variable?
14. What does it mean to initialize a variable?
15. What is an assignment statement?
16. List five rules you must take into consideration when naming variables.
17. Define the term "expression."
18. What does it mean when a programming language is "loosely typed"?
19. How does Netscape describe JavaScript in terms of its typedness?
20. What do the built-in functions `parseInt` and `parseFloat` do?
21. Describe a scenario when converting a string to a number might be necessary.
22. What does NaN mean? When are you likely to encounter it? Can you use it in a comparison? Explain.
23. What does the `typeof` operator do?
24. What is an object wrapper?
25. List three tips that you would give a new programmer to help him avoid problems with JavaScript's dynamic data typing.

Exercises

1. Which of the following are valid variable declarations or initializations? Explain why each one is or is not valid. If an item is not valid, explain how you could change it so that it is.
   a. var bean_count = 39
   b. my name = "Sam"
   c. var zipCode = document.myForm.zip.value
   d. var 1stName = "Sam"
   e. his-name = "Devan"
   f. var phoneNum = "(619)555-1212"
   g. var Tim'sNum = "(619)555-1213"
   h. var car make = "Nissan"
   i. var num3 = 88
   j. var #99 = 99
2. Evaluate each of the following expressions and specify whether the result is a string, numeric, Boolean data type, or some special value.
   a. 7 + 5
   b. "7" + "5"
   c. "7" + 5
   d. 7 + "5"
   e. "Hi " + 5
   f. 5 + 2 + " up"
   g. parseInt("7" + "5")
   h. parseInt("7") + parseInt("5")
   i. parseInt("seven" + 5)
   j. parseInt(nine)
3. List the result of each of the following typeof statements. Try to figure them out without running them in a script. Then run each statement and see how you did.
   a. typeof (-78)
   b. typeof ("Sam")
   c. typeof ("50 ways to leave your lover")
   d. typeof(parseFloat("50 ways to leave your lover"))
   e. typeof (parseInt("7up"))
   f. typeof (true)
   g. typeof (87 + "")
   h. typeof (86 + 99)
   i. typeof (count)
   j. typeof (null)
   k. typeof (parseInt("seven")
   l. typeof ()
4. Write the appropriate code to convert each of the following to a string and then back to a number.
   a. 17
   b. 181.99
   c. -97.56
   d. -87
   e. 0

5. Write the appropriate JavaScript expression to write each of the following decimal numbers in hexadecimal. (Hint: They correspond to the color values allowed in the browser-safe color palette.)
   a. 00
   b. 51
   c. 102
   d. 153
   e. 204
   f. 255
6. Specify the data type of each of the following. Assume the Web document has the following code defined before the statements:

```
<img name="MyPicture" src="images/me.jpg"
width=100 height=100>
```

   a. var num = "7"
   b. var num2 = 57.5
   c. var happy
   d. var myPic
   e. MyPicture.src
   f. MyPicture.height
   g. var val = null
   h. var truth = false
   i. var myDoc = window.document
   j. var myImage = document.MyPicture
   k. var num3 = 87
   l. var myNum = parseInt("87")
   m. var zNum = parseInt("eight")
7. Specify whether each of the following is a reserved word, a word currently used by JavaScript, a word currently used by HTML, a word currently used by CSS, or a word that is perfectly safe to use as a variable name.
   a. background
   b. backColor
   c. border
   d. checkBox
   e. client
   f. code
   g. color
   h. count
   i. data
   j. double
   k. email
   l. font
   m. java
   n. leftMargin
   o. location
   p. map
   q. max
   r. myVar
   s. name
   t. number
   u. position
   v. radioButton
   w. selection
   x. single
   y. string
   z. visitor
8. Write two JavaScript statements, one to print your name and one to print your nickname in quotation marks. Both your name and your nickname should appear in the document on the same line, like this:

   Tina Spain McDuffie "WebWoman"
9. Write a single JavaScript statement to display your favorite quote in a <blockquote> tag, within quotation marks, and written in italics. Choose a fairly long quote that

spans multiple lines. Preface it with “*Person’s name* says/said:”. Here’s an example of what the output should look like:

Arthur C. Clarke said:

“Any sufficiently advanced technology is indistinguishable from magic.”

10. Write the appropriate JavaScript statement to write the following statement to the document:

He said, “It’s not whether you win or lose, but how you play the game that counts & don’t forget that!”

Scripting Exercises

Create a folder named assignment03 to hold the documents you create during this assignment. Save a copy of the personal home page you modified at the end of Chapter 2 as home.html in the assignment03 folder.

1. Create a new document named favColor.html. Write a script that
   a. Prompts the visitor for his or her name and assigns it to a variable.
   b. Prompt the user for his or her favorite of the following colors: red, green, blue, magenta, yellow, teal, or silver.
   c. Use those variables to set the background color of the document and to welcome the user by name.
2. Create a new document named sum.html. Write a script that
   a. Prompts the user for two numbers.
   b. Then displays their sum in the following format: 10 + 15 = 25
3. Modify the following script to both declare *and* initialize each variable on a single line. Change the values to suit your own personal tastes.

```
<script language="JavaScript" type="text/javascript"><!--
      var favDrink
      var favSoda
      var favFruit
      var favSalad
      var favDinner
      var favDessert
      var favCandyBar

      favDrink = "Lemonade"
      favSoda = "Dr. Pepper"
      favFruit = "orange"
      favSalad = "Ceasar Salad"
      favDinner = "Chicken-n-Bean Burrito at Teo Leo's"
      favDessert = "Chocolate Fudge Cake Sundae"
      favCandyBar = "Butterfinger"
```

```
// -->
</script>
```

Use the modified script to write the content for the Favorite Foods section in your personal home page.

4. Modify home.html as follows:
   a. In the head of the document, declare and initialize the following variables:
      i. myName
      ii. myNickname
      iii. myGender
      iv. myBirthday
   b. Use these variables and JavaScript to complete the About Me section like this:

      Name: your name here
      Nickname: your nickname here
      Gender: male or female
      Birthday: your birthday here

   c. For an extra challenge, and to make everything line up nicely, write the information in a table. Use JavaScript to write all of the information including the tags necessary to create the table.
5. Modify home.html as follows:
   a. In the head of the document, declare and initialize the following variables:
      i. myTitle
      ii. myJobDescription
      iii. myEmployer

      Declare and initialize additional variables if you want.
   b. Use these variables and JavaScript to complete the My Job/Work section like this:

      I'm a(n) <insert myTitle here> at <insert myEmployer here>. My duties include: <insert myJobDescription here>.

      or something similar. The idea is to create some variables and use them to write information to the screen, a very common and important task performed with JavaScript.
   c. For an extra challenge, fancy your content up with HTML in your document.write statements.
6. Modify home.html as follows:
   a. Declare and initialize the following variables:
      i. myName (if not already created)
      ii. myEmail
      iii. myPhone
      iv. myWebsite

b. Remove the words "Contact Info" from the <address> tag at the bottom of the page and replace them with the following:

This site maintained by: <your name here>
Email: email@address.com Phone: (123)456-7890
<web site URL here>

or something similar. Use your variables and JavaScript to write the content. Make your email address and the Web site URL links. To do so, you'll need to use the `myEmail` and `myWebsite` variables twice each.