**Learning JavaScript**

There are many programming languages in existence today, and in this chapter, you will begin learning the basics of a programming language called JavaScript, which is by far the most common programming language used in web pages.

Although it is not possible to teach you everything there is to learn about JavaScript in one or two chapters, there are thousands of free scripts available on the Web that you can use. Therefore, the aim of this chapter is to teach you enough to start using these scripts in your web pages and to understand how they work. You should even be able to customize these scripts and write some basic scripts of your own based upon what you will learn in this and the following chapter. In addition, it will serve as a good introduction to general programming concepts.

As you will see, JavaScript gives web developers a programming language to use in web pages that allows them to perform tasks such as the following:

* Read elements from documents and write new elements and text into documents
* Manipulate or move text
* Perform mathematical calculations on data
* React to events, such as a user clicking a button
* Retrieve the current date and time from a user’ s computer or the last time a document was modified
* Determine the user’ s screen size, browser version, or screen resolution
* Perform actions based on conditions such as alerting users if they enter the wrong information into a form

**What Is Programming About?**

As you will see in this chapter, programming is largely about performing different types of *calculations* upon various types of data (including numbers, text and graphics). In all programming languages. Youcan perform tasks such as:

* Performing mathematical calculations on numbers such as addition, subtraction, multiplication, and division.
* Working with text to find out how long a sentence is, or where the first occurrence of a specified letter is within a section of text.
* Checking if one value (numbers or letters) matches another.
* Checking if one value is shorter or longer, lower or higher than another.
* Performing different actions based on whether a condition (or one of several conditions) is met. For example, if a user enters a number less than 10, a script or program can perform one action; otherwise it will perform a different action.
* Repeating an action, a certain number of times or until a condition is met (such as a user pressing a button).

These actions might sound rather simple, but they can be combined so that they become complicated and powerful. As you will see, different sets of actions can be performed in different situations different numbers of times, to create a huge variety of results.

**How to Add a Script to Your Pages:**

JavaScript can either be embedded in a page or placed in an external script file (rather like CSS). But in order to work in the browser, the browser must have JavaScript enabled. (The major browsers allow users to disable JavaScript, although very few people do.)

You add scripts to your page inside the <script> element. The type attribute on the opening <script> tag indicates what scripting language will be found inside the element, so for JavaScript you use the value text/JavaScript.

*There are several other scripting languages that do a very similar job to JavaScript (such as VBScript or Perl), but JavaScript is the main programming language used in web browsers.*

Here you can see a very simple script that will write the words “ My first JavaScript ” into the page:

<html>

<body>

<script type=”text/javascript”>

document.write(“My first JavaScript”)

</script>

</body>

</html>

In this case, we are using the write() method to add a new line of text into the web page (and the web page is represented using the document object). The text is added into the page where the script is written in the page.

Where you put your JavaScript within a page is very important. If you put it in the body of a page — as in this example — then it will run (or *execute)* as the page loads. Sometimes, however, you will want a script ready to use as soon as the page has loaded, or you might want to use the same script in several parts of the page, in which case it tends to live inside the <head> element on the page (because scripts in the head of the page load before the page is displayed). Scripts that live in the head of a page are triggered (or *called)* by an event such as when the page finishes loading or when a visitor presses the submit button on a form.

You can also write JavaScript in external documents that have the file extension .js (just in the same way that you can write external style sheets). This is a particularly good option because:

* If your script is used by more than one page you do not need to repeat the script in each page that uses it.
* If you want to update your script you need only change it in one place.
* It makes the HTML page cleaner and easier to read.

When you place your JavaScript in an external file, you need to use the src attribute on the <script> element; the value of the src attribute should be an absolute or relative URL pointing to the file containing the JavaScript. For example:

<script type=”JavaScript” src=”scripts/validation.js”> </script>

So there are three places where you can put your JavaScripts , and a single HTML document can use all three because there is no limit on the number of scripts one document can contain:

* **In the <head> of a page:** These scripts will be called when an event triggers them.
* **In the <body> of a page:** These scripts will run as the page loads.
* **In an external file:** If the link is placed inside the < head > element, the script is treated the same as when the script lives inside the head of the document waiting for an event to trigger it, whereas if it is placed in the <body> element it will act like a script in the body section and execute as the page loads.

***The <noscript> Element***

The <noscript> element offers alternative content for users who have disabled JavaScript. It can

contain any HTML content that the author wants to be seen in the browser if the user does not have JavaScript enabled.

*Strictly speaking, the W3C ’ s recommendations say that the content of this element should be displayed only when the browser does not support the scripting language required; however, the browser manufacturers have decided that it should also work when scripting is disabled.*

*Example:*

*<html>*

*<body>*

*<script src=”external.js” type=”text/JavaScript”>*

*</script>*

*<noscript> This only shows if the browser has JavaScript turned off. </noscript>*

*</body>*

*</html>*

**Starting with JavaScript**

**Writing JavaScript:**

You need to be aware of a few points when you start writing JavaScript:

* JavaScript is case - sensitive, so a variable called myVariable is different than a variable called MYVARIABLE, and both are different than a variable called myvariable .
* When you come across symbols such as ( , { , [ , `` , and ` they must have a closing symbol to match: ´ , ˝ , ] , } , and ) . (Note how the first bracket opened is the last one to be closed, which is why the closing symbols are in reverse order here.)
* Like HTML, JavaScript ignores extra spaces, so you can add white space to your script to make it more readable. The following two lines are equivalent, even though there are more spaces in the second line:

myVariable=”some value”

myVariable = “some value”

* If you have a large string, you can break it up with a backslash, as you can see here:

document.write(“My first \

JavaScript example”)

* But you must not break anything other than strings, so this would be wrong:

document.write \

(“My first JavaScript example”)

* You can insert special characters such as ˝ , ´ , ; , and & , which are otherwise reserved (because they have a special meaning in JavaScript), by using a backslash before them like so:

document.write(“I want to use a \”quote\” mark \ & an ampersand.”)

**This writes out the following line to the browser:**

**I want to use a “quote” mark & an ampersand.**

* If you have ever used a full programming language such as C++ or Java, you know they require a semicolon at the end of each line. This is optional in JavaScript unless you want to put more than one statement on a line.

**A Word About Data Types:**

By now you should be getting the idea that you can do different things with different types of data. For example, you can add numbers together but you cannot mathematically add the letter *A* to the letter *B*. Some forms of data require that you are able to deal with numbers that have decimal places (floating point numbers); currency is a common example. Other types of data have inherent limitations; for example, if I am dealing with dates and time, I want to be able to add hours to certain types of data without getting 25:30 as a time (even though I often wish I could add more hours to a day).

Different types of data (letters, whole numbers, decimal numbers, dates) are known to have different *data types*; these allow programs to manage the different types of data in different ways. For example, if you use the + operator with a string, it concatenates two strings, whereas if it is used with numbers, it adds the two numbers together. Some programming languages require that you specifically indicate what type of data a variable is going to hold and require you to be able to convert between types. While JavaScript supports different data types, as you are about to see, it handles conversion between types itself, so you never need to worry about telling JavaScript that a certain type of data is a date or a *string* (a string being a set of characters that may include letters and numbers).

There are three simple data types in JavaScript:

* **Number:** Used to perform arithmetic operations (addition, subtraction, multiplication, and division). Any whole number or decimal number that does not appear between quotation marks is considered a number.
* **String:** Used to handle text. It is a set of characters (including numbers, spaces, and punctuation) enclosed by quotation marks.
* **Boolean:** A Boolean value has only two possible values: true and false. This data allows you to perform logical operations and check whether something is true or false.

You may also come across two other data types:

* **Null:** Indicates that a value does not exist. This is written using the keyword null. This is an important value because it explicitly states that no value has been given. This can mean a very different thing from a string that just contains a space or a zero.
* **Undefined:** Indicates a situation where the value has not been defined previously in code and uses the JavaScript keyword undefined. You might remember that if you declare a variable but do not give it a value, the variable is said to be undefined (you are particularly likely to see this when something is not right in your code).

**Keywords:**

You may have noticed that there are several keywords in JavaScript that perform functions, such as break, for, if, and while, all of which have special meaning; therefore, these words should not be used in variable, function, method, or object names. The following is a list of the keywords that you should avoid using (some of these are not actually used yet, but are reserved for future use):

abstract boolean break byte case catch char class const

continue default do double else extends false final

finally float for function goto if implements import

in instanceof int interface long native new null

package private protected public return short static

super switch synchronized this throw throws transient

true try var void while with

**Starting to Program with JavaScript:**

Now that you have seen how JavaScript is able to access a document in the web browser using the DOM, it is time to look at how you use these properties and methods in scripts. As I mentioned earlier, a programming language mainly performs calculations. So, here are the key concepts you need to learn in order to perform different types of calculations:

* A *variable* is used to store some information; it ’ s like a little bit of the computer ’ s memory where you can store numbers, strings (which are a series of characters), or references to objects. You can then perform calculations to alter the data held in variables within your code.
* *Operators* perform functions on variables. There are different types of operators — for example:
* Arithmetic operators enable you to do things such as add ( + ) numbers together, or subtract ( - ) one from another (providing they are numbers).
* Comparison operators enable you to compare two strings and see if one is the same as the other, or different (for example, whether *x* is equal to *y* or whether *a* is greater than *b* ).
* *Functions* are parts of a script that are grouped together to perform a specific task. For example, you could have a function that calculates loan repayments, and when you tell the loan calculator function the information it needs (the amount of money to be borrowed, the number of years the loan will last, and the interest rate) the function will be able to return the monthly payment. Functions are objects in their own right and are very similar to things called methods; one of the key differences is that methods often belong to an object already, whereas functions are customized.
* *Conditional statements* allow you to perform different actions based upon a condition. For example, a condition might be whether a variable holding the current time is greater than 12. If the condition is true, code to write “Good Afternoon” might be run. Whereas, if it is less than 12, a different block of code saying “Good Morning” could be shown.
* *Loops* can be set up so that a block of code runs a specified number of times or until a condition is met. For example, you can use a loop to get a document to write your name 100 times.
* There are also several built - in JavaScript objects that have methods that are of practical use. For example, in the same way that the document object of the DOM has methods that allowed you to write to the document, the built - in JavaScript date object can tell you the date, time, or day of the week.

The following section looks at these key concepts in more detail.

**Variables:**

Variables are used to store data. To store information in a variable, you can give the variable a name and put an equal sign between it and the value you want it to have. Here is an example:

userName = “Bob Stewart”

The variable is called userName and the value is Bob Stewart. If no value is given, then its value is *undefined*.

The script can access the value of the variable using its name (in this case userName ). It can also change the value.

You can create a variable, but not store anything with it by using the var keyword; this is known as *declaring* a variable (unlike some other languages, you do not have to declare a variable before you can use it, although it is commonly considered good practice to do so).

var userName

There are a few rules you must remember about variables in JavaScript:

* They must begin with a letter or the underscore character.
* Variable names are case - sensitive.
* Avoid giving two variables the same name within the same document as one might override the value of the other, creating an error.
* Do not call two variables the same name, but use different cases to distinguish them (e.g., username and userName ) as this is a common source of confusion later.
* Try to use descriptive names for your variables. This makes your code easier to understand (and will help you debug your code if there is a problem with it).

**Assigning a Value to a Variable:**

When you want to give a value to a variable, you put the variable name first, then an equal sign, and then on the right the value you want to assign to the variable. You have already seen values being assigned to these variables when they were declared a moment ago. So, here is an example of a variable being assigned a value and then the value being changed:

userName = “Bob Stewart”

userName = “Robert Stewart”

userName is now the equivalent of Robert Stewart.

**Lifetime of a Variable:**

When you declare a variable in a function, it can be accessed only in that function. (You will learn about functions shortly.) After the function has run, you cannot call the variable again. Variables in functions are called *local variables*.

Because a local variable works only within a function, you can have different functions that contain variables of the same name (because each is recognized by that function only).

*If you declare a variable using the* var *keyword inside a function, it will use memory up only when the function is running, and once the function has finished it will not take up any memory.*

If you declare a variable outside a function, all the functions on your page can access it. The lifetime of these variables starts when they are declared and ends when the page is closed. Local variables take up less memory and resources than page - level variables because they require only the memory during the time that the function runs, rather than having to be created and remembered for the life of the whole page.

**JavaScript Output**

JavaScript can "display" data in different ways:

* Writing into an HTML element, using innerHTML
* Writing into the HTML output using document.write()
* Writing into an alert box, using window.alert()
* Writing into the browser console, using console.log()

**Operators:**

The operator itself is a keyword or symbol that does something to a value when used in an *expression*. For example, the arithmetic operator + adds two values together.

The symbol is used in an expression with either one or two values and performs a calculation on the values to generate a result. For example, here is an expression that uses the multiplication operator:

area = (width \* height)

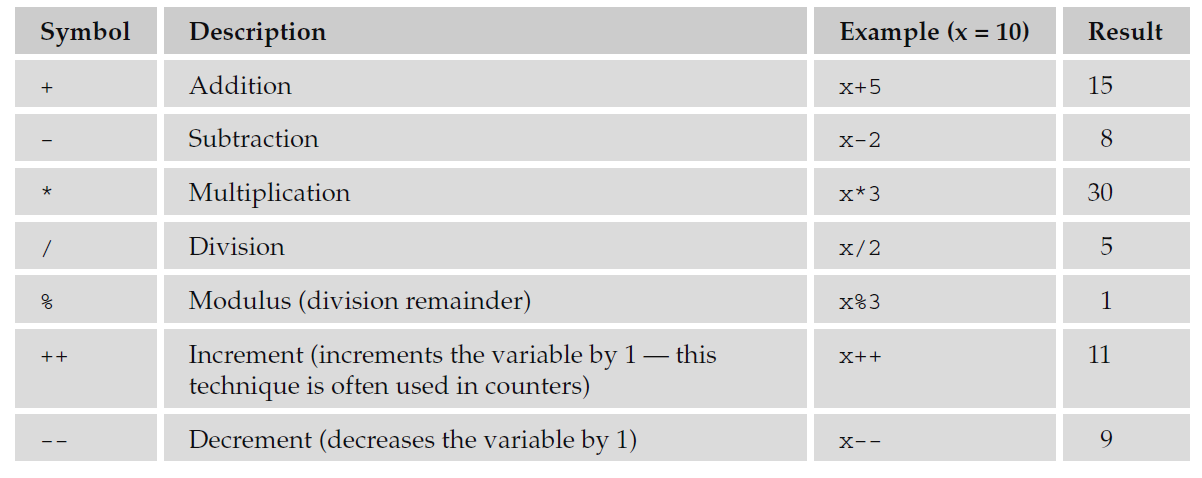
An expression is just like a mathematical expression. The values are known as *operands*. Operators that require only one operand (or value) are sometimes referred to as *unary operators*, while those that require two values are sometimes called *binary operators*.

The different types of operators you will see in this section are:

* Arithmetic operators
* Assignment operators
* Comparison operators
* Logical operators
* String operators

**Arithmetic Operators:**

Arithmetic operators perform arithmetic operations upon operands. (Note that in the examples in the following table, x = 10.)



**Assignment Operators:**

The basic assignment operator is the equal sign, but do not take this to mean that it checks whether two values are equal. Rather, it’ s used to assign a value to the variable on the left of the equal sign, as you have seen in the previous section, which introduced variables.

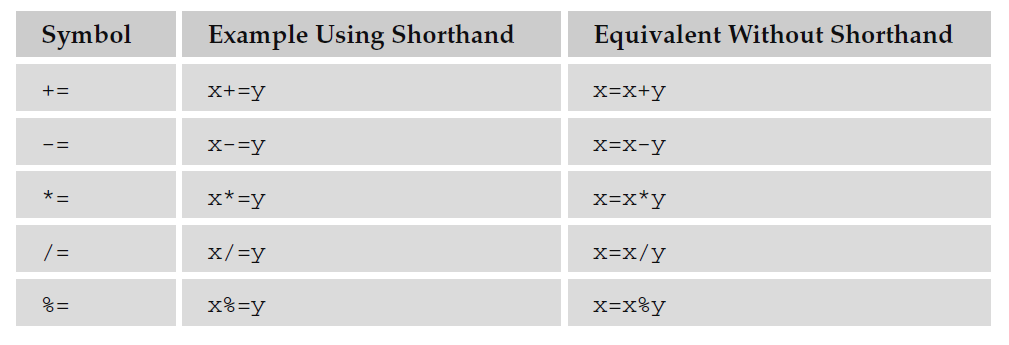
The basic assignment operator can be combined with several other operators to allow you to assign a value to a variable *and* perform an operation in one step. For example, take a look at the following statement where there is an assignment operator and an arithmetic operator:

total = total – profit

This can be reduced to the following statement:

total -= profit

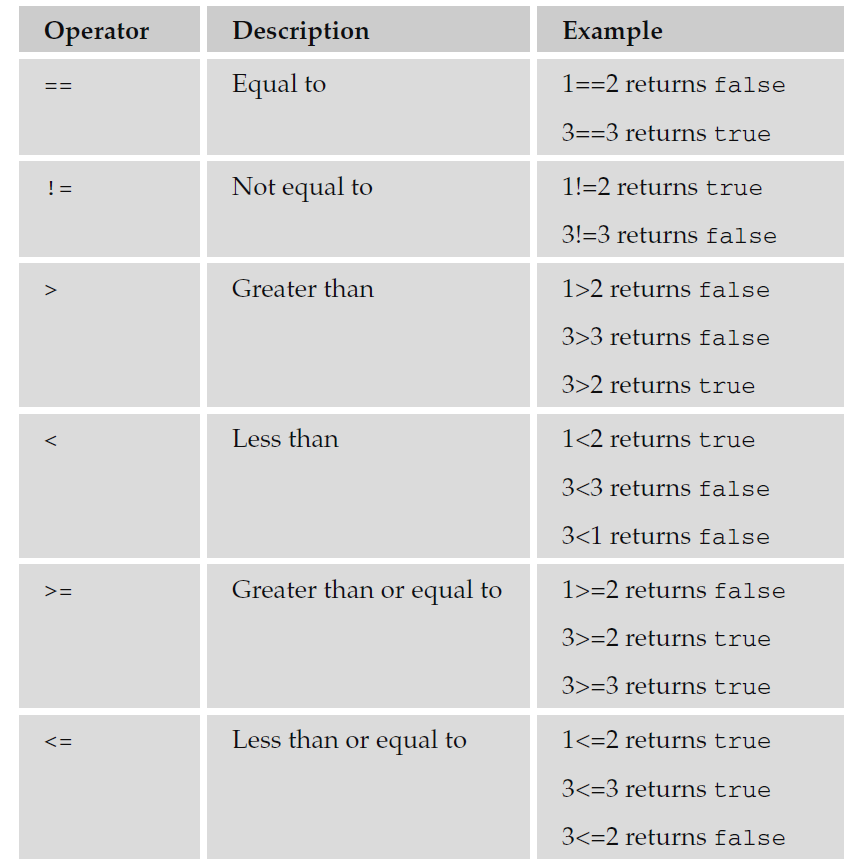
While it might not look like much, this kind of shorthand can save a lot of code if you have a lot of calculations like this (see table that follows) to perform.



**Comparison Operators:**

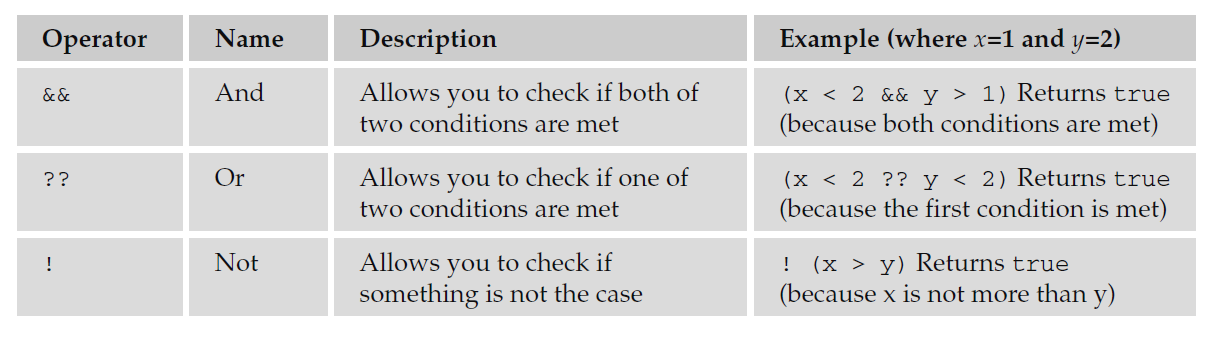
As you can see in the table that follows, comparison operators compare two operands and then return either true or false based on whether the comparison is true or not.

Note that the comparison for checking whether two operands are equal is two equal signs (a single equal sign would be an assignment operator).



**Logical or Boolean Operators:**

Logical or Boolean operators return one of two values: true or false. They are particularly helpful when you want to evaluate more than one expression at a time.



The two operands in a logical or Boolean operator evaluate to either true or false. For example, if x=1 and y=2, then x < 2 is true and y > 1 is true. So, the following expression:

(x < 2 & & y > 1)

returns true because both of the operands evaluate to true (you can see more examples in the right-hand column of this table).

**String Operator (Using + with Strings):**

You can also add text to strings using the + operator. For example, here the + operator is being used to add two variables that are strings together:

firstName = “Bob”

lastName = “Stewart”

name = firstName + lastName

The value of the name variable would now be Bob Stewart. The process of adding two strings together is known as *concatenation*.

You can also compare strings using the comparison operators you just met. For example, you could check whether a user has entered a specific value into a text box.