Variables: An Introduction

A computer stores data, among other things. The term "data" sounds intimidating, but it really isn't. Data is simply information. On the computer, data can be numbers, or words, or pictures, or sounds, or movies, or anything.

When you program in any imperative programming language, you will need to instruct the computer when to store data, and when to transfer data, and when to delete data, etc. Data, however, can be much more than mere information; it can also be references to actual elements of the web page or the web browser itself. You'll see what I mean as we go along.

In imperative programming languages, such as JavaScript, data is stored in VARIABLES. A variable is really just a container for information, for data.

In JavaScript, ANYTHING can be stored in a variable. A number or a word could be stored in a variable. A picture or a sound file could be stored in a variable. Even a reference to an actual browser window or an entire HTML page could be stored in a variable in JavaScript, allowing you to open or close (or otherwise manipulate) that browser window (for instance), or even alter an individual HTML element in that browser window, via the appropriate JavaScript variable. After we have explored some more basic JavaScript programming concepts, I will show you how variables and actual browser elements work together. Until I have given you some more theoretical information about programming, however, none of my explanations of the practical applications would make any sense, so please bear with me.

Think of a variable as if it were a box. You take data and you put it into a box (a variable). You can also replace the data in one of these variable boxes with new data.

A variable MUST have a name; otherwise, you'll have no way to refer to that variable in a manner which the computer (or YOU) can understand. In JavaScript, names ARE case-sensitive (along with everything else), so SPELLING COUNTS! Variable names follow the standard naming conventions we have discussed in earlier modules, and should usually begin with a lower-case letter.

In the following example, I will create a variable called "george". I am going to fill george with a number, the number 1.

Here is a conceptual picture of george:

As you can see, george, the variable, is a box with a name, containing the number 1. In JavaScript, you will use variables to store all data, whether it be numbers, or words, or pictures, or whatever.

Declaration and Initialization

When creating variables in an imperative programming language, there is a two-step process: 1) declaration (declaring a variable), and 2) initialization (initializing a variable). Again, these technical words sound intimidating, but they're not.

When you DECLARE a variable, you tell the computer that that variable exists; you also tell it what that variable's name is. When you INITIALIZE a variable, you fill that variable with data for the FIRST time (initial = first), setting it to its initial, or beginning, state.

Computers are stupid. If you don't tell them that something exists, they can't figure it out for themselves; that's why you have to declare a variable.

In JavaScript, you would declare "george" (from the above example) like this:

var george;

To declare a variable in JavaScript, you must first state the JavaScript keyword, "var" (which tells the computer that you want to create something of type "var" or variable), followed by a space, followed by the name that you want to call that variable (in this case, the name is george); the command is finished off with a semi-colon to end the line of code.

Once you have declared a variable, you may refer and make changes to that variable simply by stating its name and performing OPERATIONS upon it. Most operations in JavaScript (including addition (+), subtraction (-), multiplication (\*), and division (/), among others) are performed with a handful of symbols, many of which will be already familiar to you; I will talk more about these OPERATORS a little later.

In JavaScript, I could initialize the already-declared variable, george, using the following syntax:

george = 1;

In the above example, I have initialized the variable, george, setting it equal to the number 1; I have done this using the "gets" operator, which is represented by the equals (=) sign.

The "gets" operator (=) replaces the contents of whatever is on its left-hand side with whatever is on its right-hand side. In the above example, george "gets" 1; this means that the "gets" operator takes the number 1 from its right-hand side and sticks that value into george on its left-hand side, replacing george's contents; since george has never contained any data, this process INITIALIZES george, setting him to a starting value of 1. Again, I have ended my command, my line of code, with the semi-colon end-of-line marker; every complete command in JavaScript must end its line with a semi-colon.

A COMMAND is an instruction or set of instructions for the computer, telling it to do something (or multiple somethings). Every time you come to the end of a command in JavaScript (which will end a line of code), you must mark the end of that line with a semi-colon, as in the above examples.

I could also declare and initialize the variable, george, in a single step using the following syntax:

var george = 1;

First, I declare george to exist by using the var keyword, then I use the "gets" operator to initialize george to contain the number 1. Although it is possible to combine these two processes (declaration and initialization) together into one command as in the above example, it is important for you to understand that these are two separate actions, and that there will be times when you will NEED to separate variable declaration and initialization for one reason or another.

A Note on Semi-Colons

JavaScript is derived from the programming language called "C". All C-based languages end lines of code with semi-colons; a semi-colon (;) is the end-of-line or end-of-command marker. You have already seen a similar use of the semi-colon in CSS, where a semi-colon marks the end of a property definition (i.e. **font-size:24px;**).

Note: Older code written in JavaScript does NOT always use a semi-colon to mark the end of a line of code. Theoretically, JavaScript, under many circumstances, may count carriage-return characters as a replacement for semi-colon characters; this does not always work properly, however. I cannot recommend the omission of semi-colon characters when coding commands in JavaScript, as it opens you up to unexplained errors due (I assume) to bugs in the JavaScript interpreters.