**Typed and Untyped Languages and Simple Variable Operations**

In JavaScript, variables may contain ANY type of data. When you declare a variable, you only need to state the keyword var and the name of the variable; this creates a generic variable. Once you have a variable declared, you can stick a number or a string or a boolean or an object into the variable, and change the data type of the variable at will.

Example:

var george = 1;

george = "Hi";

In the above example, I have set george equal to the number 1. In the second line of code, I have replaced the contents of george with a string, "Hi". I have changed george's data type from a number to a string. This kind of action is possible because JavaScript is an "untyped" language. In an untyped language, a variable may be of any data type, and an individual variable's data type may be changed at any time.

Most programming languages, like C++ and Java, are "typed" languages. In a typed language, a given variable may only ever be ONE data type. For instance, in C/C++, you would create an "int" variable type, and that variable could only contain an integer, or you would create a "string" variable type, and that variable could only ever contain a string.

Example (C/C++):

int george = 1;

string fred = "Hi";

float ethel = 1.052;

In the above example in C/C++, I have declared and initialized three different variables. The variable george is an integer (int) and may only ever contain an integer. The variable fred is a string, and may only ever contain a string. The variable ethel is a float, and may only ever contain a number with decimal places. In typed languages like C/C++, variable data types remain fixed for the duration of a program.

As I mentioned earlier, even though JavaScript is an untyped language, and even though you can change a variable's data type on-the-fly in JavaScript, variables can only be a SINGLE data type at any given time, whether that type be a number, a string, a boolean, an object, or null/undefined.

Here are some more JavaScript examples of variables being declared, initialized, and otherwise operated upon:

var kate = "hi";

var louie = 10.25;

var josie = louie;

var sam = 200;

var answer = louie + josie + sam;

kate = answer;

In order of the examples: the variable kate now contains a string value, "hi". The variable louie now contains a number. The variable josie has been set equal to louie, which means that a copy of louie's contents have been placed inside of josie; both josie and louie, then, are equal to 10.25. The variable sam now contains a number as well. The variable called answer has been set to the sum of louie and josie and sam, which would be 10.25 + 10.25 + 200, or the number 220.5. Note: answer is just a random name, picked because it makes sense in English; I could have named it anything. The variable kate (which has already been declared), then, has had its string content of "hi" replaced with the numerical content of the variable answer, which changes kate's data type from a string to a number.

The following example is WRONG:

louie + josie + sam = answer;

Even though the above example looks like math operations that you might perform on paper, the above syntax is backwards. The variable answer needs to contain the sum of louie + josie + sam, which is only possible if the variable answer is on the LEFT-HAND side of the "gets" (=) operator (since the "gets" operator always replaces the contents of the left-hand variable with whatever is on the right-hand side). Left-to-right orientation, then, is important in JavaScript, as well as case-sensitivity!

Example (CORRECT):

var answer = louie + josie + sam;

The following example is ALSO WRONG:

var answer = louie + JOSIE + Sam;

Earlier, I initialized the variables josie and sam using all lower-case letters in their names. Since JavaScript is case-sensitive, JOSIE and Sam represent different, non-existent, undefined variables. The variable answer in the above example, then, would end up containing non-valid information, producing a bug and a JavaScript error in your code.

Again, I apologize for the abstractness of these examples, but you must understand the basics of JavaScript variable syntax before you can do anything at all with the language. I hope that you can see, though, that handling these variables and performing simple mathematical operations upon them is fairly straightforward as long as you understand the basic logic behind their syntax.

Of course, you can do a lot more than mathematical operations with JavaScript; we'll get to a few simple, practical things before the end of this module.