Introduction to Variables

Now let's take a deeper dive into **variables**. We've introduced the concept to the variable, but what is a variable? And how do we use them in programming?

Variable Defined

By definition, variables are named pieces of data, stored in memory, that can be written to, read from, and modified. In the example that we created before, we made a variable called `example`. In this case, the piece of data was the string "coffee", it was named example. We could write to it by assigning it a value of "coffee". We could read from it by doing a `console.log`. And we can also modify variables once they've been created.

Symbolic

Variables are symbolic, that means that any variable can stand for any possible value of a single type. And we've talked about data types before. Variables are subject to change, you don't know from one point to another in a program, what a variable might stand for, or even if it's been assigned a value. And a variable can only refer to a single value at any given time.

Why Use Variables?

So, why do we use variables in programming? Let me give you an example. What if we had a program that did a `console.log(2+2)` and we've seen before it that JavaScript interprets 2 plus 2 as 4. If we run this program, we can see that JavaScript outputs the value of 4. But what if we wanted to write a program that could `console.log` the values of two different numbers being added together, without knowing in advance what those two numbers might be?

```
var first;
var second;

first = 1;
second = 2;

console.log(first);
console.log(first + second);
first = 3;
console.log(first + second);
```

By using variables, we've made this code much more flexible because it doesn't care what the values are. Every time you run it, it will figure out what values `first` and `second` have been set to, and do the calculation appropriately. As we get into more complex programming structures, you're going to see this comes in handy.

Characteristics of a Variable

So, what are the characteristics of a variable?

- Variables all have names, and that's how you reference the data.
- Variables have a type, which is the kind of information stored in that particular variable. That type can change as we've discussed, depending on what you've assigned the value of that variable to be.

- Variables have a value, and the value is the data stored in the variable.
- Variables also have a memory location, depending on the compiler you're using, JavaScript will store the value of that variable somewhere in memory. You don't need to know about where it's being stored, but you do need to know that it's being stored somewhere.
- Variables also have a scope, when you're writing a complex program in JavaScript, sometimes there will be places in the program where the variable works, and other places where it hasn't been defined yet, or where it's not understood or recognized. The scope of a variable helps you figure out where you can use that variable.
- Variables also have a lifespan, before a variable has been defined, it doesn't exist.
- Variables can also be destroyed, which ends their lifespan and they can no longer be accessed.

Steps for Using Variables

- Before you use a variable you have to **declare** it, that is you have to tell the program that you're going to be using a variable of a certain name, so that it knows that it exists.
- You have to set that that variable to a value.
- You're going to read the value.
- You might also change the value, you might delete the variable.

There are other things that you could do, but these are the basics for how you use a variable.

If you're going to use a variable in a particular scope, you want to declare that variable at the top of that scope.

```
var first;
first = 1;
console.log(first);
```

I've declared a variable called `first`, then I've assigned it a value. And lastly I've read that value.

Why Declare Variables?

So, why do we have to declare variables before we use them?

No matter how many JavaScript programs are running simultaneously, if you declare a variable on the global scope, every single program that's running simultaneously, will be able to access that variable by name, and if you make changes to a variable that another program has created, that other program isn't going to know that your program and the other program aren't the same program. They're going to treat that globally scoped variable the same way. By declaring a variable, you define what scope that variable is available in, so that you don't interfere with other programs.

A lot of web pages simultaneously load multiple scripts from different sources, and if two different programmers who wrote two different programs both created a variable called `first`, and they're both using it and changing it, you don't know if the value of `first` that you're using is yours, or the one that came from the other program.

The reason that we declare variables first, as soon as we create a scope in JavaScript, is because of Variable Hoisting. Basically once you've created a scope inside of JavaScript, you want all of the variables that you're going to use declared right at the beginning of that scope. Because no matter where you declare them inside of that scope, the first time the compiler runs into that piece of code, it's going to go and declare all of the variables that are needed in that scope before running any of the code. Since the ECMAScript 5 version of JavaScript, which has been in most browsers since 2014, a standard has emerged to add a `use strict;` string to the beginning of programs. This forces compilers not to allow variables not to be used unless they've been declared with the `var` keyword before they were used.

It's also possible to declare and initialize a variable at the same time, by using the `var` keyword giving the variable a name, and assigning it a value all in one line.

Declare and Initialize a Variable at Once

```
var first = 1;
```

So as you see we don't need a separate line to assign a variable to a value.

When you declare a variable, whether you want to assign it to a value at that time or not, depends on how you're going to be using it: sometimes you have no idea what you're variable is going to be set to. And sometimes you want to set it to a default value, which you may change later in the script. If that's the case, assigning it at the time that you declare it is a good idea.