Functions in JavaScript

This is lesson five of Introduction to Programming in JavaScript. I'm going to take you into the realm of **functions** in JavaScript. So what is a function and why would you want to use one?

Calling a Block of Code

First of all we've shown you a few different ways of calling blocks of code. For example, we've shown you iterators. Float control can branch to different blocks of code. But, what if you just want to call a certain block of code that might exist somewhere else in your program?

Well, what if you could give a block of code a name? By giving a block of code a name, you'd be able to call it whenever you wanted to. It'd be kind of like creating a new keyword in JavaScript. That's exclusively for use with your own code. That's kind of what a function is.

A function lets you name a block of statements and call it whenever you want to.

```
function sayHi() {
  console.log("Hi.");
}

sayHi();
// "Hi"
```

As you remember previously we've shown you that methods followed by parentheses executes some code, in this case the code that lives within the function.

Declaring and Using Functions

Declaring and using functions in JavaScript is pretty straight forward. Functions can be defined either before or after the code that uses them as long as their available in the same scope. Functions can make use of any variables that existed when the function was called and they can make changes to those variables as well, so be careful.

```
var color;

console.log(color); // undefined
changeColor(); // function defined below
console.log(color); // "green"
changeColor();
console.log(color); // "blue"
changeColor();
console.log(color); // "red"
changeColor();
console.log(color); // "green" (etc.)

function changeColor() {
  switch(color) { // has access to color
    case "red":
      color = "blue";
      break;
```

```
    case "green":
      color = "red";
      break;
    default:
      color = "green";
  }
}
```

When JavaScript runs, the very first thing it does is goes through the entire set of code available to it, finds all of those functions and makes them available to the code that you're going to be running. The other thing to keep in mind is that the `color` variable, which lives outside of the function, was being modified inside the function, and it was changed by that function.

Result of Calling a Function

When you want to call a function, you need to call the name of the function followed by parentheses. If you don't use the parentheses what JavaScript does is that it returns the code that made up that function. With the parentheses that function is actually executed.

When you execute a function, you also have the option of returning a value.But if you don't specify a return value for a function the return value of that function will be `undefined`.

```
var color;

console.log(changeColor);
// shows the code of the function
console.log(changeColor());
// undefined (but the function is executed)
console.log(color);
// "green"

function changeColor() {
  switch(color) {
    case "red":
      color = "blue";
      break;
    case "green":
      color = "red";
      break;
    default:
      color = "green";
  }
}
```

So even though a function may not actually return a result, as long as it's executed it will perform the statements inside of the block of code that its executing.

Specifying a Return Value

You can specify a return value for a function. As a matter of fact, you can only specify one return value for any given function although you can use conditional logic inside of a function to have multiple places where different return values might come back. Each execution of a function can only return one value. As we've demonstrated, you must actively return a value from a function or that function will return `undefined`.

```
var color;

console.log(changeColor);
// shows the code of the function
console.log(changeColor());
// "green"
console.log(color);
// "green"

function changeColor() {
  switch(color) {
    case "red":
```

```
      color = "blue";
      break;
    case "green":
      color = "red";
      break;
    default:
      color = "green";
  }
  return(color);
}
```

It's a good practice to have a return value for every function so that you have a way of defining that the function has concluded and testing the results of that function.

Return Terminates Execution

The `return` statement in a function, once it gets there, will stop execution of the function. Anything that you put in a function beyond that `return` statement will not be executed. There can only be one value returned for every execution of a function, and the `return` statement can be used to stop the execution of a function much the same way that a `break` statement can be used to break out of the execution of a loop.

```
var color;

console.log(color); // undefined
color = returnColor(); // execute the function
console.log(color); // "green"
color = returnColor();
console.log(color); // "red"

function returnColor() {
  switch(color) {
    case "red":
      return("blue"); // return stops the function
    case "green": // no break is needed
      return("red");
    default:
      return("green");
  }
  console.log("This will never display");
}
```

When you're writing functions, you can forget sometimes that the code that comes after a `return` statement won't be executed. Keep that in mind so that you don't end up writing code that's never going to get called.