Functions

A function is a set of statements that perform a certain task. You can think of it like making a class in CSS that groups a bunch of code together to apply it to one thing.

The format for declaring a function is the same every time:

`function functionName() {}`

Overview

```
let number = 0;
function functionNameGoesHere() {
    // Write your code that does something. Can be whatever you want, but here's a simple example.
    number++;
}
```

So, we've declared our function and wrote what we want it to do. However, it won't actually run any of the code until we tell it to. To run this function, all you have to do is type the name of the function and two parenthesis after it:

```
functionNameGoesHere();
```

This is referred to as `calling` a function. Any time this function is called, it will run all of the code inside of it. In this case, all it will do is increase the number variable by 1.

Parameters and Arguments

Sometimes you will want to give your function some information or extra options. In these cases, you can add `parameters` to the function. You can think of parameters like variables specifically for that function, and you can name them whatever you want. These parameters go in between the parenthesis when you declare the function.

```
function add(firstNumber, secondNumber) {
    let result = firstNumber + secondNumber;
}
```

This function needs two numbers to add together, so we add those two parameters in the function declaration. When we run the function, we will need to supply it with those two numbers, otherwise it's not going to know what two numbers to add.

```
add(1, 2);
```

This is called passing `arguments` to a function. This will put the number 1 in place of the firstNumber parameter, and the number 2 in place of the secondNumber parameter. So, it would effectively look like this:

```
function add(1, 2) {
    let result = 1 + 2;
}
```

Scope

Let's take a look at the add function from earlier, along with the function call:

```
function add(firstNumber, secondNumber) {
    let result = firstNumber + secondNumber;
}

add(1, 2);
```

This works, but let's say we need to access the result variable later to display it to the user. This is where `scope` comes into play: scope refers to where a variable is declared and how it can be accessed. Generally speaking, there are two types of scope:

Global Scope

These are variables declared outside of any function or block. These can be accessed anywhere in the code; there are no restrictions on them.

Example:

```
// Since this is declared outside of any function or block, we can access it
anywhere.
let result = 0;

function add(firstNumber, secondNumber) {
    // We can update the result variable since it's global
    result = firstNumber + secondNumber;
}

add(1, 2);
```

Block Scope

These are variables declared within a function or block (between two curly brackets). These can only be accessed within that block or function.

Example:

```
function add(firstNumber, secondNumber) {
    // This variable is declared within the 'add' function, meaning we can't access it anywhere else.
    let result = firstNumber + secondNumber;
}

add(1, 2);

// What do you think will happen if we try to access the result variable outside of its scope?
result = 3;

// This will work, because now we're re-declaring it outside of the function, and now it has global scope. However, this is not a good idea. Why?
let result = 5;
```

Returning Data From Functions

What if we have a calculation in a function, and we absolutely need to access the result of that calculation? We could define the variable outside the function, do the calculation and assign the value to the variable within the function. But, there's often a better way using the `return` keyword. Return will stop the function and return a value from the function. Take a look at this code:

```
function add(firstNumber, secondNumber) {
    return firstNumber + secondNumber;
}

let result = add(firstNumber, secondNumber);
```

Since we added a return to the function, it will add the two numbers together and output the result from the function. However, we still need some place to store that returned value, so we set result equal to the function call. This will store the firstNumber + secondNumber calculation in the result variable. If we didn't add the return keyword, the function wouldn't output anything even though the calculation still happens.

Methods

In JavaScript, methods are just functions that are built into the language. There are a ton of these, but some that you already know are:

`console.log()` - logs something to the debug console. The argument tells it what to log.

`alert()` - puts an alert box on the screen that displays a message. The argument tells it what to tell to the user.

`prompt()` - similar to alert, but also allows the user to enter a response that is returned from the method.

```
let number = 3;
console.log(number);

alert("You are being alerted");

// This asks the user to enter a number, and their response is returned from the
method
// Since it's returning their response, we can store it in a variable called
userInput
let userInput = prompt("Enter a number");
```

Other methods that we've already used, such as `array.push()`, is also a method. Generally, anything that has parenthesis after it is either a function or a method, and is usually colored yellow in VSCode.

You can find all methods on MDN or W3Schools:

https://developer.mozilla.org/en-US

https://www.w3schools.com/jsreF/default.asp