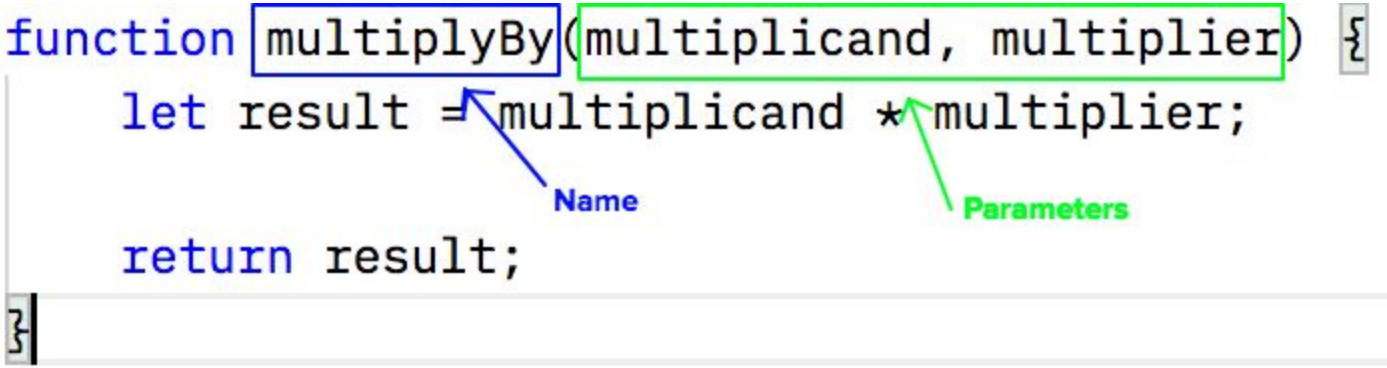
Module 3 - Lecture 7

JavaScript Functions

Named Functions

Named Function

- camelCase naming convention
- No return type
- No data types on parameters
- Returning a value is optional



Parameters

- When calling a function, supplying parameter values is optional.
  - They will default to undefined if not supplied.

```
function multiplyBy(multiplicand, multiplier) {
  let result = multiplicand * multiplier;

  return result;
}
```

```
multiplyBy()
```

Default Parameters

- I can supply default values for parameters.
  - The default value will be used if a value is not supplied by the caller.

```
function multiplyBy(multiplicand = 0, multiplier = 0) {
    let result = multiplicand * multiplier;

    return result;
}
```

An unknown number of parameters

- When I call a function, parameters values that I pass are stored in **an array-like object named arguments**.

```
function concatAll() { // No parameters defined, but we still might get some
    let result = '';
    for(let i = 0; i < arguments.length; i++) {
        result += arguments[i];
    }
    return result;
}
```

An unknown number of parameters

- I can also write a function that accepts a variable number of parameters. JavaScript will store these in an array.

```
function myFun(a,  b, ...manyMoreArgs) {
  console.log("a", a)
  console.log("b", b)
  console.log("manyMoreArgs", manyMoreArgs)
}

myFun("one", "two", "three", "four", "five", "six")

// Console Output:
// a, one
// b, two
// manyMoreArgs, ["three", "four", "five", "six"]
```

Question:
If a parameter's value is modified within a function, does that modification persist when I return from that function?

Answer:
It depends on the data type of the parameter.

Changes to object parameters will persist.

This includes:

- Object
- Arrays
- Date
- and more

```
function willObjectChange(arr, obj) {
    arr.push(5);              // add 5
    obj.firstName = 'Bob';    // Change firstName
}

let a = [2, 3];     // array with two elements

let b = {           // object with one property
  firstName: 'Walt'
};

willObjectChange(a, b);

console.log(a);
console.log(b);
```

```
▶ (3) [2, 3, 5]
▶ {firstName: "Bob"}
```

Changes to more primitive types will not.

This includes:

- Number
- String
- Boolean

```
function willPrimitiveChange(num, str, bool) {
    num = num + 5;            // add 5
    str = str + ' Changed!';  // concatenate ' Changed!'
    bool = !bool;             // flip it to the opposite
}

let a = 1;
let b = 'Test';
let c = true;

willPrimitiveChange(a, b, c);

console.log(a);
console.log(b);
console.log(c);
```

```
1
Test
true
```

Anonymous Functions & Arrow Functions

```
function (a, b) {
    return a * b;
}
```

```
(a, b) => {
    return a * b;
}
```

Traditional Functions vs Arrow Functions

```
1  // Traditional Function
2  function (a){
3    return a + 100;
4  }
5
6  // Arrow Function Break Down
7
8  // 1. Remove the word "function" and place arrow between the argument and opening body
9  (a) => {
10   return a + 100;
11 }
12
13 // 2. Remove the body brackets and word "return" -- the return is implied.
14 (a) => a + 100;
15
16 // 3. Remove the argument parentheses
17 a => a + 100;
```

Things to note:

- if your arrow function has 0 or more than 1 parameter, parentheses are required.
- if your arrow function has more than 1 statement, surrounding curly braces are required.

Array Methods

- Several of the methods of an Array in JavaScript require a **callback function**.

- A callback function is a function passed into another function that the called function invokes.

- This is done to simplify common tasks with Arrays.

Using array method with a callback

```
const myArray = [1, 2, 5, -1, 4];

myArray.every((item) => item > 0);
```

Without a callback

```
const myArray = [1, 2, 5, -1, 4];

function isEverythingPositive(arr) {
    for(let i = 0; i < arr.length; i++) {
        if(arr[i] <= 0) {
            return false;
        }
    }

    return true;
}

isEverythingPositive(myArray);
```

How this works...

```
function every(callback) {
    for(let i = 0; i < myArray.length; i++) {
        if(!callback(myArray[i])) {
            return false;
        }
    }

    return true;
}
```

Other Array Methods

- **forEach**
  - Loop over each element of an array
- **map**
  - Map each element of an array to a new value.
- **filter**
  - Keep elements that meet the filter criteria. Remove others.
- **reduce**
  - Reduce the array to a single value. Summing an array is one example use case.
- **some / every**
  - Return true if some / every element meets a condition.

```
map([🐮, 🌭, 🐔, 🌽], cook)
=> [🍔, 🍟, 🍗, 🍿]

filter([🍔, 🍟, 🍗, 🍿], isVegetarian)
=> [🍟, 🍿]

reduce([🍔, 🍟, 🍗, 🍿], eat)
=> 💩
```

QUESTIONS?