Java
Module 3 - Day 7

JavaScript Functions

Today’s Objectives

- Named Functions
  - Parameters
  - Optional Parameters
  - Default Values
- Arguments Variable
- Array Functions
  - forEach
  - map
  - filter
  - reduce

JavaScript Functions

- Defined with the **function** keyword, followed by a **name**, followed by parentheses **()**
- Parenthesis may include parameter names
  - Parameters are NOT required
- Functions ***may*** **return** a value
  - Functions are not required to return anything
- The **()** operator **invokes** the function
  - Accessing a function without () returns the function definition
- Arguments Object (parameters get weirder)
  - The arguments object - JavaScript | MDN

Lecture Code: JavaScript Functions

...up to “Arrow Functions”

JavaScript Arrow Functions

- Also called “Anonymous Functions”
  - Why? They aren’t named.

```
(argument1, argument2, ... argumentN) => {
  // function body
}
```

- Array Function and Return Type
  - **forEach** - executes a function one for each array element
  - **map** - returns a new array
  - **filter** - returns a new array
  - **reduce** - returns the resulting value

array.forEach()

[Array.prototype.forEach() - JavaScript]

forEach

You've seen the `forEach()` function before. It acts like a `for` loop, running a passed in anonymous function for every element of an array:

```
let numbers = [1, 2, 3, 4];

numbers.forEach( (number) => {
    console.log(`This number is ${number}`);
});
```

array.map()

Array.prototype.map() - JavaScript

map

`map` acts like `forEach`, but it returns a new array using the return value of the anonymous function as the values in the new array:

```
let numbersToSquare = [1, 2, 3, 4];

let squaredNumbers = numbersToSquare.map( (number) => {
    return number * number;
});

console.log(squaredNumbers);
```

array.filter()

[Array.prototype.filter() - JavaScript]

`filter`

`filter` takes an anonymous function, runs each element through it, and returns a new array. If the function returns `true`, the element is kept in the new array. If the function returns `false`, the element is dropped from the new array:

```
let numbersToFilter = [1, 2, 3, 4, 5, 6];

let filteredNumbers = numbersToFilter.filter( (number) => {
    // Only keep numbers divisible by 3
    return number % 3 === 0;
});

console.log(filteredNumbers);
```

array.reduce()

[Array.prototype.reduce() - JavaScript]

The reduce() method executes the callback function once for each assigned value present in the array, taking four arguments: Accumulator, currentValue, currentIndex, array

```
[0, 1, 2, 3, 4].reduce(function(accumulator, currentValue, currentIndex, array) {
  return accumulator + currentValue
})
```

The callback would be invoked four times, with the arguments and return values in each call being as follows:

callback iteration	*accumulator*	*currentValue*	*currentIndex*	*array*	return value
first call	0	1	1	[0, 1, 2, 3, 4]	1
second call	1	2	2	[0, 1, 2, 3, 4]	3
third call	3	3	3	[0, 1, 2, 3, 4]	6
fourth call	6	4	4	[0, 1, 2, 3, 4]	10

array.reduce()

[Array.prototype.reduce() - JavaScript]

You can also provide an Arrow Function instead of a full function. The code below will produce the same output as the code in the block above:

```
[0, 1, 2, 3, 4].reduce( (accumulator, currentValue, currentIndex, array) => accumulator + currentValue )
```

If you were to provide an `initialValue` as the second argument to `reduce()`, the result would look like this:

```
[0, 1, 2, 3, 4].reduce((accumulator, currentValue, currentIndex, array) => {
    return accumulator + currentValue
}, 10)
```

callback iteration	*accumulator*	*currentValue*	*currentIndex*	*array*	return value
first call	10	0	0	[0, 1, 2, 3, 4]	10
second call	10	1	1	[0, 1, 2, 3, 4]	11
third call	11	2	2	[0, 1, 2, 3, 4]	13
fourth call	13	3	3	[0, 1, 2, 3, 4]	16
fifth call	16	4	4	[0, 1, 2, 3, 4]	20

Lecture Code:
Tutorial - Arrow Functions