

Module 3-6

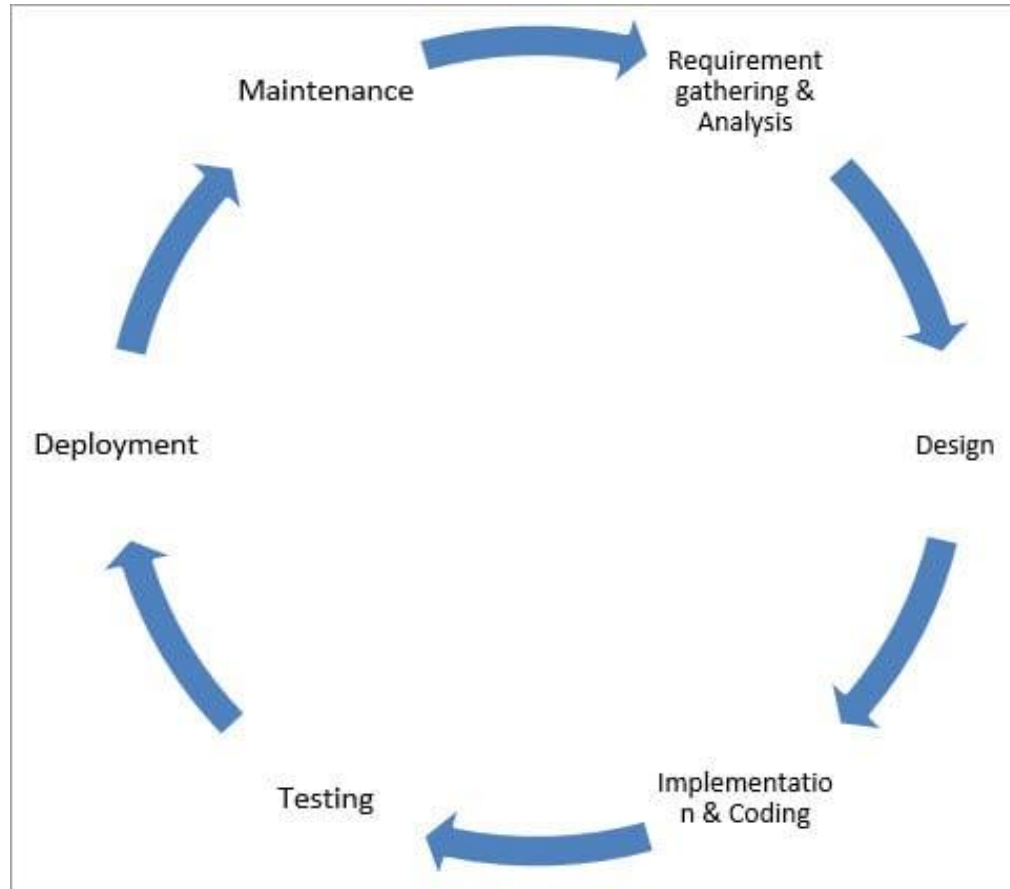
JS Functions

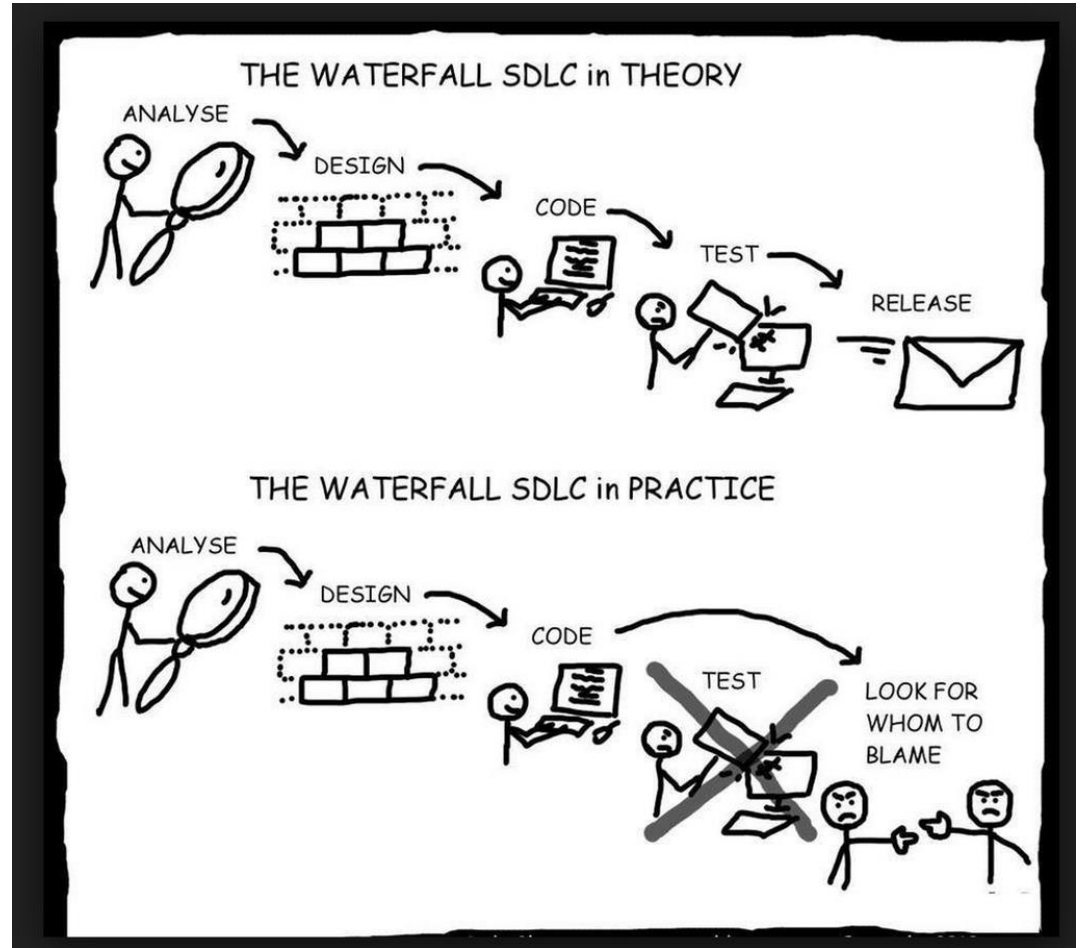
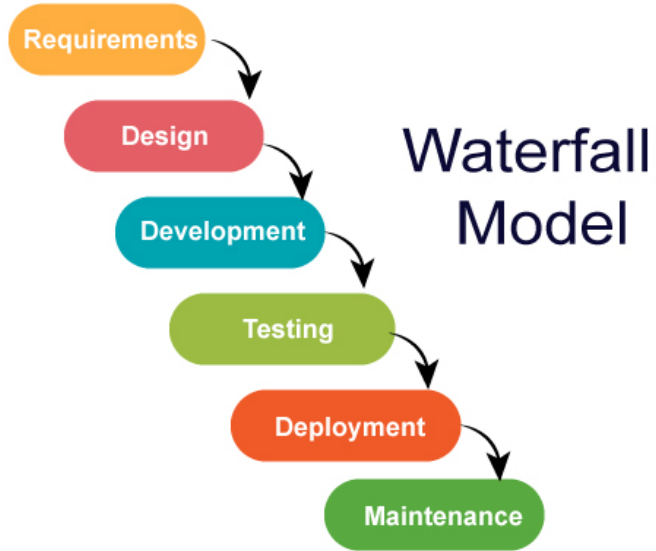
Objectives

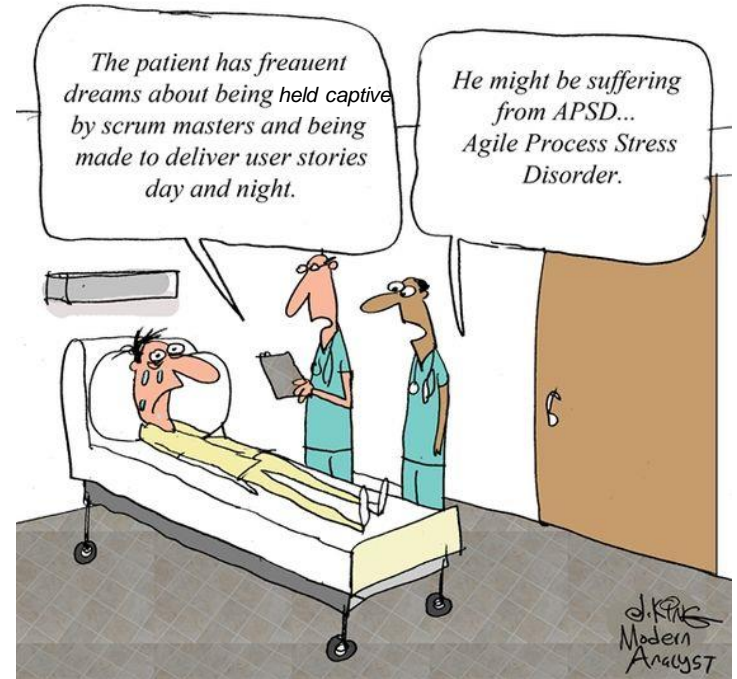
- Functions introduction
- Named functions
- Function parameters
 - Optional Parameters
 - Parameter default values
 - Arguments variable
- Anonymous functions
- Array functions
- Function documentation



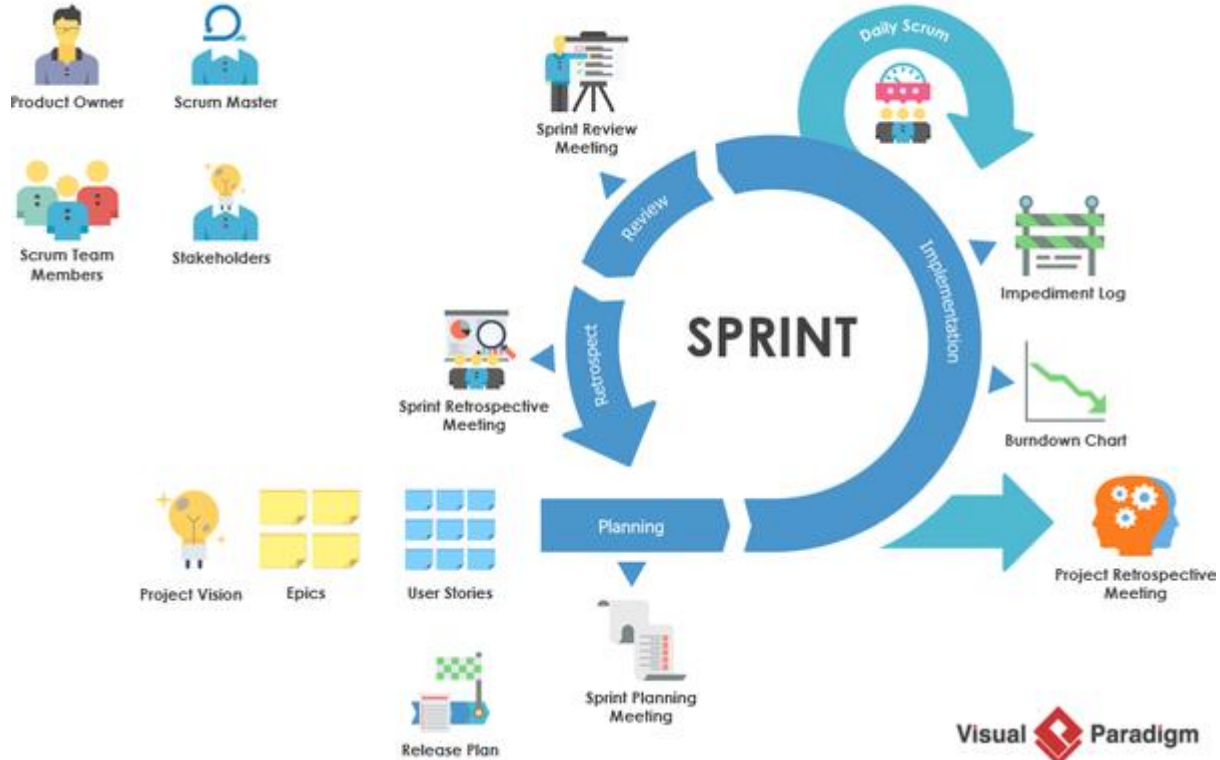
SDLC







The Agile – Scrum Framework

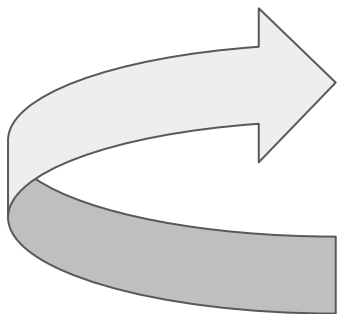


JS Functions

- The function is the primary organizational unit of JS, as opposed to classes in Java.
- Logically, we can also think of javascript functions as being similar to methods within Java classes - they have input parameters, can have outputs you can return etc.

JS Functions Defining and Calling

Here is an extremely simple named function definition, and a call to that function:



```
// A function is defined:  
function doSomething() {  
    console.log('Function is like a method');  
}  
  
// A method is called:  
doSomething();
```

Note that unlike Java, no return data type is defined.

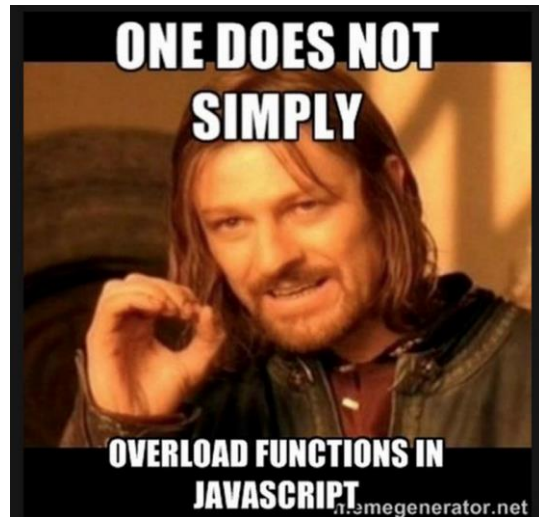
JS Function argument

JavaScript can take in arguments, here is an example:

```
// A function is defined:
function doSomething(a, b) {
  console.log(a+b);
}

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

// A function is called:
doSomething(3, 9); // 12
let result = returnSomething(9,2);
console.log(result); // 18
```



JS Function argument, weirdness

Unlike Java, passing in a parameter is optional, regardless of how a function is defined! You can also assign an optional value for when no argument for the parameter is passed in:

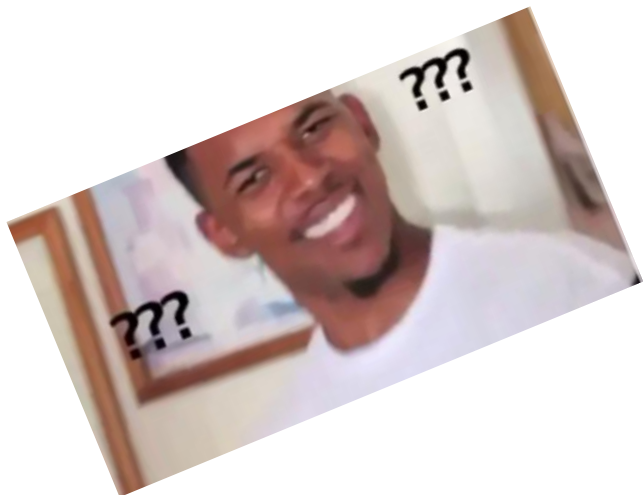
```
function returnSomething(a = 2, b = 5) {  
    return a * b;  
}  
  
let result = returnSomething();  
console.log(result); //10
```



The arguments parameter

```
function longestString() {  
  let longest = '';  
  for (let i = 0; i < arguments.length; i++) {  
    if (arguments[i].length > longest.length) {  
      longest = arguments[i];  
    }  
  }  
  return longest;  
}
```

```
> longestString('hi', 'this', 'supercalifragilisticexpialidocious', 'world');
```



Let's write some functions



Anonymous Functions

An anonymous function is one that does not have a name. Thus far, all the functions we've seen have a name following the form "function." Consider the following:

```
let listOfNumbers = [2, 3, 7];  
let evensOnly = listOfNumbers.forEach( (num) => (console.log(num * 2)) ); // 4 6 14
```

The anonymous function is highlighted in red, note that this function has an input (num), which represents each element of the array, it will print out that element multiplied by 2.

Anonymous Functions Uses

Anonymous functions are generally used in situations requiring a callback function.

A callback function is a function that is passed as an argument to another function.

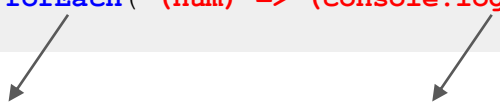
We will be looking at some examples of functions that require callback functions, and how we can use anonymous functions to make the code less verbose.



For Each with Anonymous Function

The For Each function iterates through the elements of the array. We can define an anonymous function to define what action will be undertaken during each iteration:

```
let listOfNumbers = [2, 3, 7];  
let evensOnly = listOfNumbers.forEach( (num) => (console.log(num * 2)) ); // 4 6 14
```



Call the forEach method.

Use an anonymous function to tell the forEach what to do during each iteration of the loop.

For Each without Anonymous Function

The preceding code could have been written without an anonymous function:

Without Anonymous Function:

```
let instructions = function multiplyByTwo(num) {  
  console.log(num * 2);  
}  
  
let listOfNumbers = [2, 3, 7, 76, 91];  
let evensOnly = listOfNumbers.forEach( instructions );
```

Compare the above with what we wrote on the previous screen:

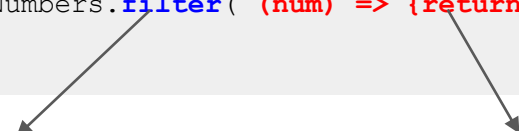
With Anonymous Function:

```
let listOfNumbers = [2, 3, 7];  
let evensOnly = listOfNumbers.forEach( (num) => (console.log(num * 2)) ); // 4 6 14
```

Filter with Anonymous Function

The filter function creates a new array from an existing one, provided that the element meets certain conditions.

```
let listOfNumbers = [2, 3, 7, 76, 91];  
let evensOnly = listOfNumbers.filter( (num) => {return num%2 === 0})  
console.log(evensOnly);
```



Here is the
filter method
being called.

Here we have an
anonymous function telling
the filter function how to
filter

Filter without Anonymous Function

Again, some comparing and contrasting. You can certainly define a function and not do it anonymously:

```
let instructions = function filterInstructions(num) {  
  
    if (num%2 ==0) {  
        return true;  
    }  
    return false;  
}  
  
let listOfNumbers = [2, 3, 7, 76, 91];  
let evensOnly = listOfNumbers.filter( instructions )  
console.log(evensOnly);
```

Without Anonymous Function:

```
let listOfNumbers = [2, 3, 7, 76, 91];  
let evensOnly = listOfNumbers.filter( (num) => {return num%2 === 0})  
console.log(evensOnly);
```

With Anonymous Function:

Map with Anonymous Function

The map function takes an array and performs some kind of operation on each row. The key point is that you still have the same number of elements.

Consider the example, we just saw, that of multiplying each element in an array by two:

```
let listOfNumbers = [2, 3, 7, 76, 91];  
let doubleNumbers = listOfNumbers.map((n) => n *2);  
console.log(doubleNumbers);  
//[4, 6, 14, 152, 182]
```

Reduce with Anonymous Function

The reduce method takes an array of numbers and makes one number out of the whole process. Simplest example? How about add every element in an array:

```
let listOfNumbers = [2, 3, 7, 76, 91];  
let doubleNumbers = listOfNumbers.reduce( (runningTotal, currentValue) => runningTotal  
+ currentValue);  
console.log(doubleNumbers); //179
```

Objectives

- Functions introduction
- Named functions

```
// A function is defined:  
function doSomething() {  
    console.log('Function is like a method');  
}  
  
// A method is called:  
doSomething();
```

Objectives

- Functions introduction
- Named functions
- Function parameters
 - Optional Parameters
 - Parameter default values
 - Arguments variable

```
function returnSomething(a = 2, b = 5) {  
    return a * b;  
}
```

```
let result = returnSomething();  
console.log(result); //10
```

```
function longestString() {  
    let longest = '';  
    for (let i = 0; i < arguments.length; i++) {  
        if (arguments[i].length > longest.length) {  
            longest = arguments[i];  
        }  
    }  
    return longest;  
}
```

Objectives

- Functions introduction
- Named functions
- Function parameters
 - Optional Parameters
 - Parameter default values
 - Arguments variable
- Anonymous functions

```
function callAnonFunction() {  
  let multiply = (x, y) => {  
    let product = x * y;  
    return product;  
  }  
  console.log(multiply(5, 6));  
}
```


Objectives

- Functions introduction
- Named functions
- Function parameters
 - Optional Parameters
 - Parameter default values
 - Arguments variable
- Anonymous functions
- Array functions

```
function playWithForEach() {  
  let myArray = [6, 7, 8, 9];  
  myArray.forEach((double) => {  
    console.log(`${double} is ${double * 2} when doubled`);  
  });  
}
```

Objectives

- Functions introduction
- Named functions
- Function parameters
 - Optional Parameters
 - Parameter default values
 - Arguments variable
- Anonymous functions
- Array functions
- Function documentation

```
/**
 * Takes an array and, using the power of anonymous functions, generates
 * their sum.
 *
 * @param {number[]} numbersToSum numbers to add up
 * @returns {number} sum of all the numbers
 */
function sumAllNumbers(numbersToSum) {
  return numbersToSum.reduce();
}
```

Let's write some anonymous functions!

