Call Stack, Execution Context and Higher Order Functions

Higher Order Functions

- A higher order function is a function that takes another function as an argument.
- Example:

```
function add(a,b) {
  return a+b;
}

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

function calculate(a,b,fn) {
  return fn(a,b);
}
```

- The function ***add()*** takes two parameters, adds the values and returns the sum.
- The function ***multiply()*** takes two parameters, adds the values and returns the product.
- The function ***calculate()*** takes three parameters and the third parameter is **a function.**
- And it returns **that function** (invoked).
- So the ***calculate()*** function is a higher-order function.
- So we can use the ***calculate()*** function to either add or multiply two values by passing in the **add()** or multiply function as the third parameter.
- Example:

```
calculate(2, 3, add)
```

- The above code will output 5.

Execution context

- It's a fancy way of saying "the environment in which the JavaScript code is running."
- The execution context is created by the JavaScript engine when you write code and run the script.
- The creation of the execution context consists of two steps:
    - The creation phase:
        - The first step is when the JavaScript engine stores the variables and function declarations.
    - The execution phase:
        - The second step is when the engines assign values to the variables and execute the function calls.

Call Stack

- It is a data structure that keeps track of the functions that are being run.
- It has two methods: ***pop()*** and ***push().***
- It's a last in first out data structure (LIFO).
- This means the last function to be pushed onto the stack is the first to be popped off the stack.
- Example:

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

function calculate(a,b,fn) {
  return fn(a,b);
}


calculate(2, 3, multiply)
```

- When the above code is run, the first function to be pushed to the stack will be the ***calculate()*** function.
- And since the calculate function is a higher order function, the next function to be pushed to the stack will be the ***multiply()*** function.
- The call stack will look something like:

```
multiply(2,3)
calculate(2,3,add)
```

- When the ***multiply()*** function returns, it is removed from the stack.
- The only function on the stack will be the ***calculate()*** function.
- And the ***calculate()*** finally returns, it is also removed from the stack.
- The call stack will now be empty and will remain empty until another function is invoked.