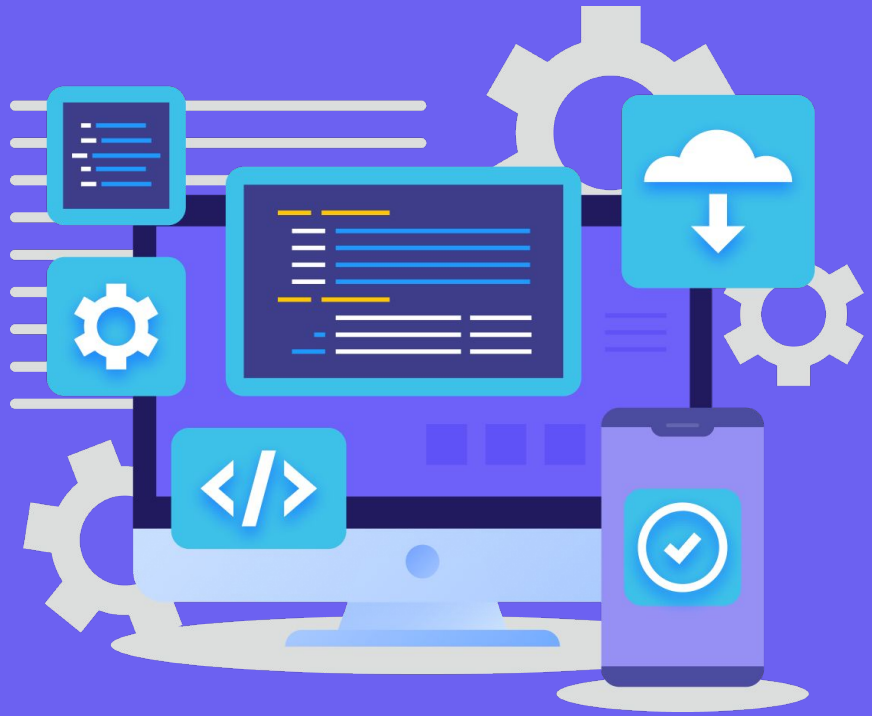
Side Effects of Functions, Scopes and Bindings, Closures, Intro to Recursion, Higher Order Functions and Abstraction, Composability



Relevel
by Unacademy

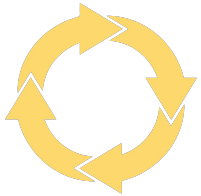
Concepts:

Side Effects of Functions

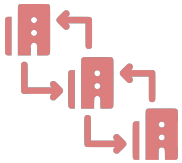
Scopes

Binding

Closures



Intro to Recursion



Higher Order Functions and Abstraction

Composability

Side Effects of Functions

Side effect is a change in the state of the program/application that is reflected outside the called function other than its return value.

```
// returns square of given number

function getSquare(input){
    return input*input;
}

console.log(getSquare(5)); // 25
console.log(getSquare(5)); // 25
```

```
function getSquare(input){
    console.log(input*input); // 25
}

getSquare(5);
```

Scope

Scope refers to the current context of your code. It can be global or local.

```
// Lexical scope

// Global scope
function getName(){
    // Local scope of getName()
    // name isn't accessible here
    function getNewName(){
        // Local scope of getNewName()
        const name = "Jack";
        console.log("Name: " + name);
    }

    getNewName(); // Name: Jack

    console.log(name);

}

getName(); // ReferenceError: name is not defined
```

Closures

It can be defined as a scenario where the inner/child function has access to the scope (all variables and functions) of the outer/parent function even after the parent function has already been executed.

```
// Closures

const greet = function(name){
    const text = 'Arigato ' + name;

    return function(){
        console.log(text);
    }
}

greet('Zoro')(); //Arigato Zoro
```

Binding in Javascript

Binding is the concept of making our function bound to an object of a certain scope.

```
const nakama1 = {
    name: 'Luffy',
    place: 'Foosha Village',
    getBio: function(){
        console.log(this.name + ' lives in ' + this.place)
    }
}

const nakama2 = {
    name: 'Zoro',
    place: 'Shimotsuki Village',
    getBio: function(){
        console.log(this.name + ' lives in ' + this.place)
    }
}

const nakama3 = {
    name: 'Sanji',
    place: 'Baratie Restaurant',
    getBio: function(){
        console.log(this.name + ' lives at ' + this.place)
    }
}

const myFunc1 = nakama1.getBio.bind(nakama1);
myFunc1(); // Luffy lives in Foosha Village

const myFunc2 = nakama2.getBio.bind(nakama2);
myFunc2(); // Zoro lives in Shimotsuki Village

const myFunc3 = nakama3.getBio.bind(nakama3);
myFunc3(); // Sanji lives at Baratie Restaurant
```

Intro to Recursion

It will be like using the same function again and again until some condition satisfies and return the output to the parent function.

```
countDownFrom = (number) => {
    if (number === 0) {
        return;
    }

    console.log(number);
    countDownFrom(number - 1);
}

countDownFrom(5);
// 5
// 4
// 3
// 2
// 1
```

Higher Order Functions and Abstraction

If a function can operate on other functions in either ways:

1. By taking functions as arguments.
2. By returning them.

is known as Higher order functions.

```
filter = (array, onBasis) => {

let passed = [];

for (let element of array) {
    if (onBasis(element)) {
    passed.push(element);
    }
}

return passed;
}

intArray = [1, 2, 3, 4, 5, 6,
console.log(filter(intArray, integers => integers % 2 == 0));
```

(parameter) integers: any

```
map = (array, mapping) => {

    let mapped = [];

    for (let element of array) {
      mapped.push(mapping(element));
    }

    return mapped;
 }

 intArray = [1, 2, 3, 4, 5];
 console.log(map(intArray, integers => integers * integers));
```

Composability

```
average = (array) => {
    return array.reduce((a, b) => a + b) / array.length;
}

let intArray = [1, 2, 3, 4, 5];
console.log(Math.round(average(
    intArray.filter(integer => integer % 2 == 0).map(evenInteger => evenInteger * evenInteger))));
```

Practice/HW

1. Program to demonstrate the use of this and bind() with closures.
2. Program to find mean, median, mode, and standard deviation using composability and higher-order functions.

Thank you