# Functional Programming

By: Mohammad Halaweh

# 01 What is Functional Programming

Programming paradigm or coding style designed to handle pure functions. This paradigm is totally focused on writing more compounded and pure functions.

# 02 Functional Programming is Declarative

- **Imperative Programming**
  programming style that we specify the program logic, by describing the flow control

- **Declarative Programming**
  programming style that we specify the program logic, without describing the flow control

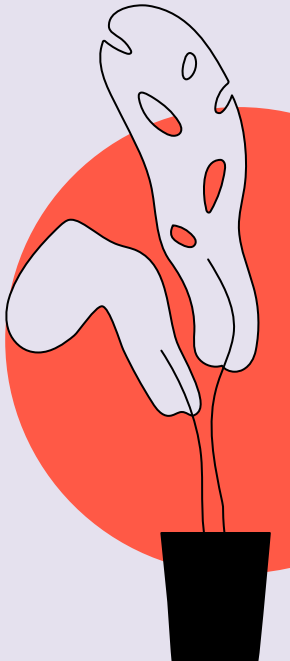
# Examples of Imperative and Declarative

```
1  const name = "Mohammad";
2  const Greeting = "Hi,"
3  console.log(Greeting, name); // Hi, Mohammad
```

Imperative

Declarative

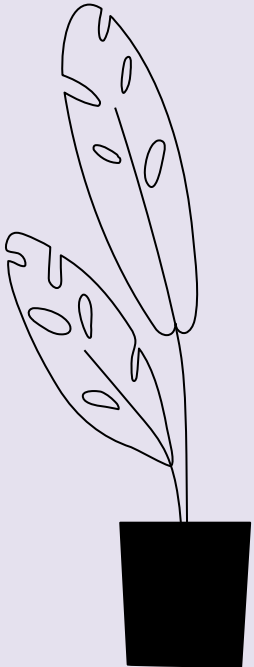
```
1  const Greeting = (name) => {
2      return "Hi, " + name;
3  };
4
5  console.log(Greeting("Mohammad")); // Hi, Mohammad
6
7
```



# 03 Pure Functions

Simple and reusable, they completely independent of the outside state (global variables), easy to refactor, test and debug.

Pure function is a function which given the same input, will always return the same output.



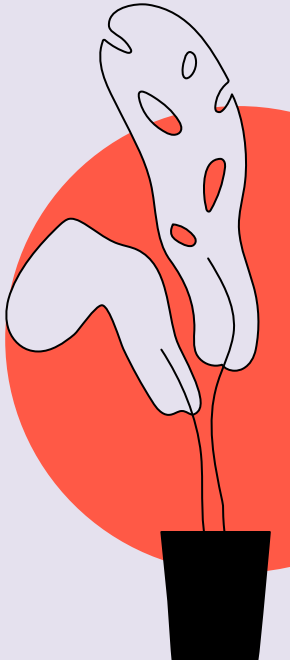
# Examples of Pure and Not Pure Functions

```
1  const add = (x, y) => {
2    return x + y;
3  };
4
5  add(2, 5); // 7
6
```

Pure

Not Pure

```
1  let counter = 0;
2
3  const incCount = (value) => {
4    return (counter += value);
5  };
6
7
```



# 04 Higher Order Functions

Functions that take other functions as inputs, or functions that return functions as its output.
(Functions can be inputs or outputs).

# Examples of Higher Order Functions

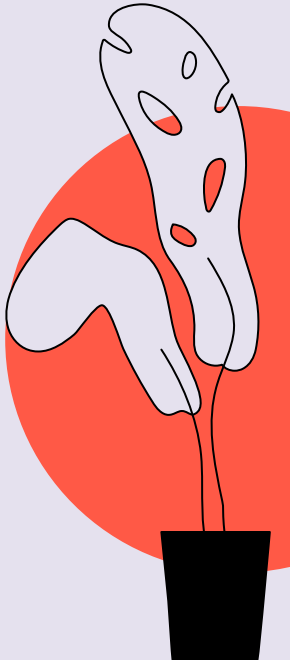
Q: Suppose this given array **arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]**

```
function filterOdd(arr) {
    const filteredArr = [];
    for (let i=0; i < arr.length; i++) {
        if (arr[i] % 2 !== 0) {
            filteredArr.push(arr[i]);
        }
    }
    return filteredArr;
}
console.log(filterOdd(arr));

// Output:
// [ 1, 3, 5, 7, 9, 11 ]
```

```
function filterEven(arr) {
    const filteredArr = [];
    for (let i=0; i < arr.length; i++) {
        if (arr[i] % 2 == 0) {
            filteredArr.push(arr[i]);
        }
    }
    return filteredArr;
}
console.log(filterEven(arr));

// Output:
// [ 2, 4, 6, 8, 10 ]
```



# Examples of Higher Order Functions

Q: Suppose this given array **arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]**

```
function filterFunction(arr, callback) {
    const filteredArr = [];
    for (let i=0; i < arr.length; i++) {
        callback(arr[i]) ? filteredArr.push(arr[i]) : null;
    }
    return filteredArr;
}
```

```
function isEven(x) {
    return x % 2 === 0;
}
```

```
function isOdd(x) {
    return x % 2 != 0;
}
```

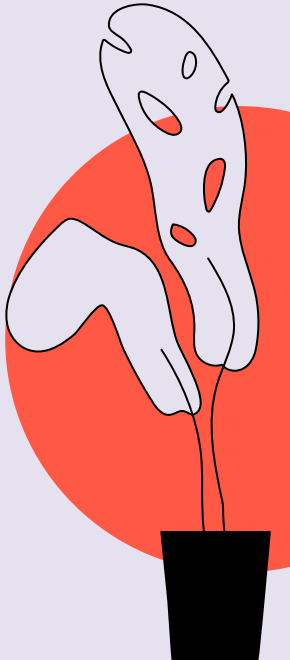
```
function isGreaterThanFive(x) {
    return x > 5;
}
```

# Examples of Higher Order Functions

```
function mackAdjectifier (adjective){
    return function(string){
        return (adjective + " " + string);
    }
}

const coolifier = mackAdjectifier('cool')
console.log(coolifier('presentation'));

// Output : cool presentation
```



# 05 Functional Programming in React

React uses the functions to make the components, these functions are pure functions.

```
function Header(props) {
    return (
        <h1>{props.text}</h1>
    )
}
```

# Thanks!

Any Questions ?