Functional Programming

By: Hamzeh Saleh

01 What is Functional Programming

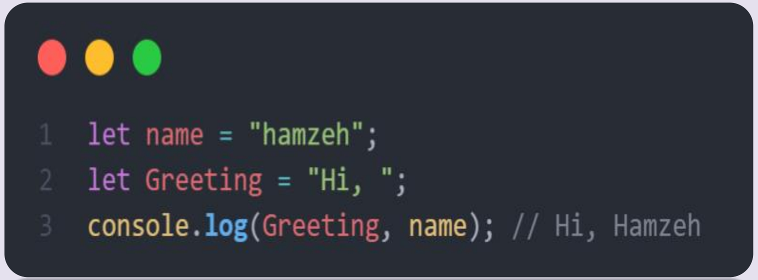
Programming paradigm or coding style designed to handle pure functions. This paradigm is totally focused on writing more compounded and pure functions.

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

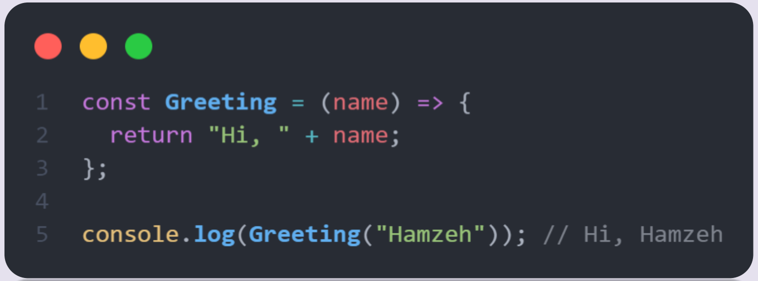
Examples of Imperative and Declarative



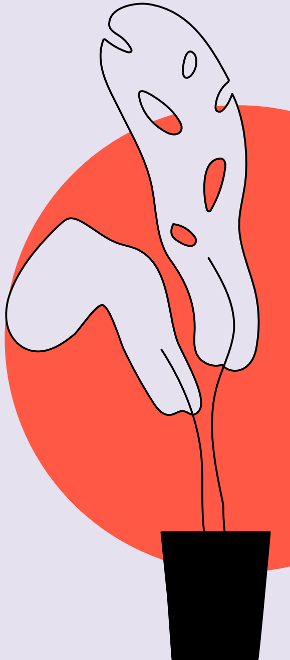
```
let name = "hamzeh";
let Greeting = "Hi, ";
console.log(Greeting, name); // Hi, Hamzeh
```

Imperative

Declarative



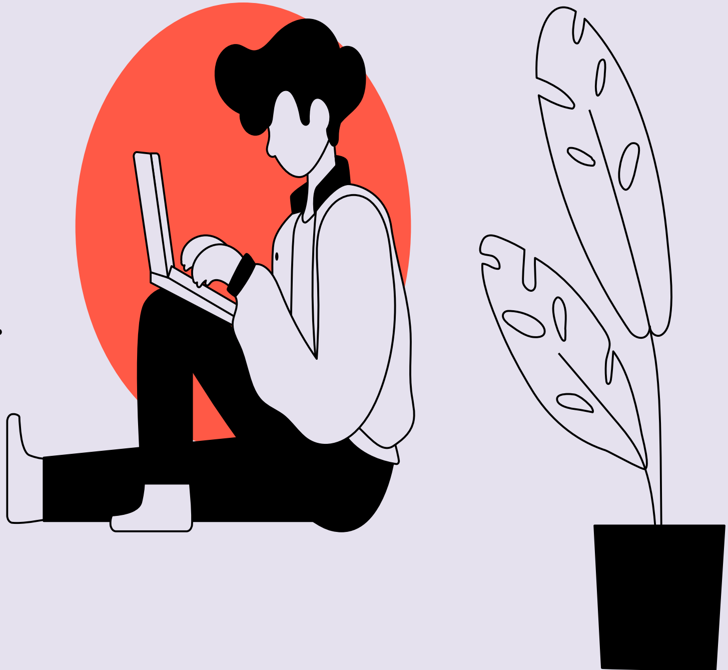
```
const Greeting = (name) => {
  return "Hi, " + name;
};

console.log(Greeting("Hamzeh")); // Hi, Hamzeh
```



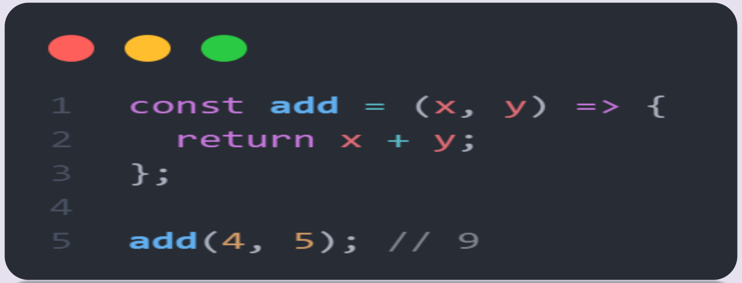
03 Pure Functions

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.



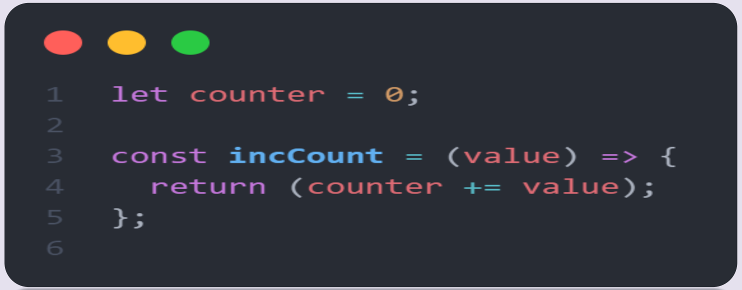
Examples of Pure and Not Pure Functions



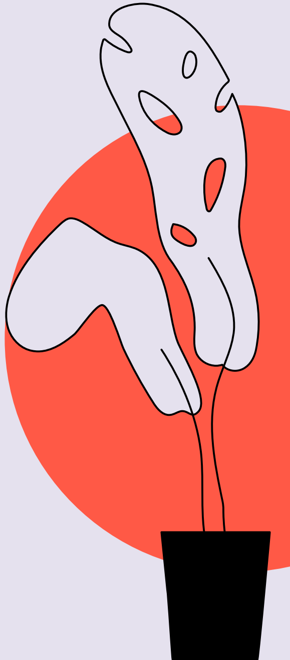
```javascript
const add = (x, y) => {
  return x + y;
};

add(4, 5); // 9
```

Pure

Not Pure



```javascript
let counter = 0;

const incCount = (value) => {
  return (counter += value);
};

```

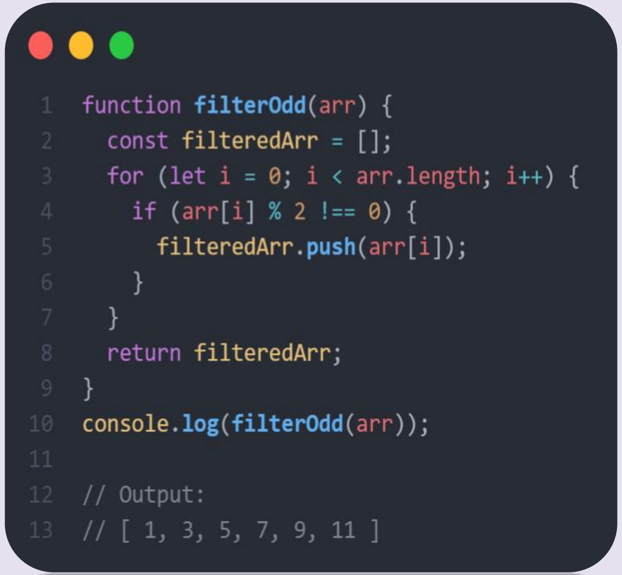


04 Higher Order Functions

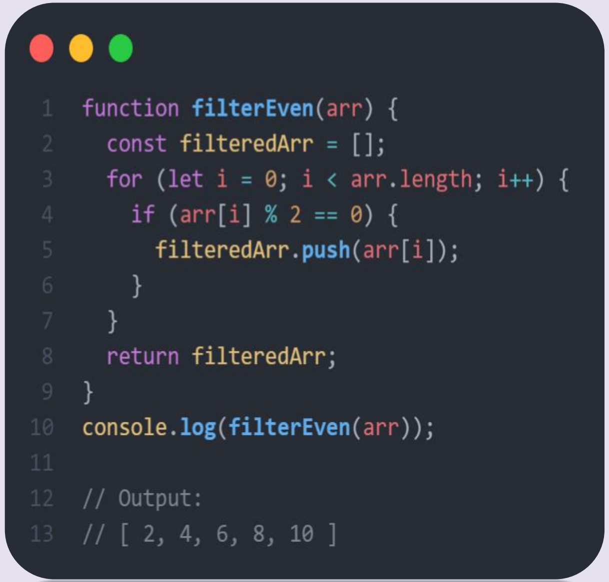
Functions that take other functions as inputs, or functions that return functions as its output.
(Functions can be inputs or outputs).

Examples of Higher Order Functions

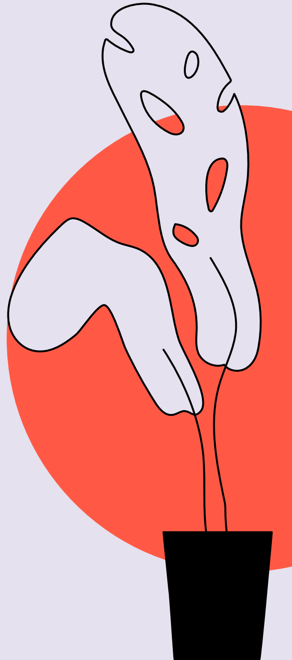
Q: Suppose this given array **arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]**



```javascript
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 ]
```

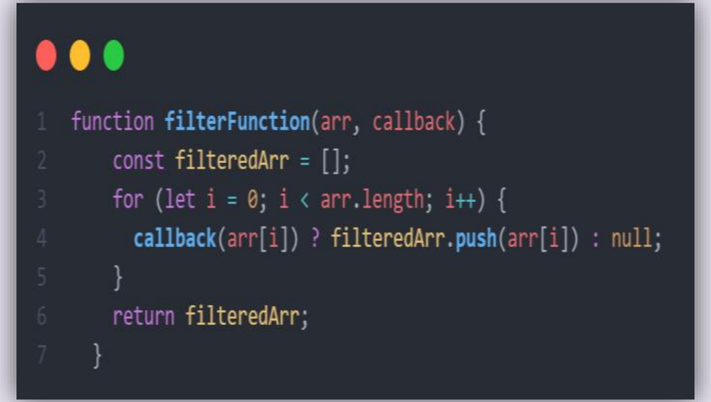


```javascript
function filterEven(arr) {
  const filteredArr = [];
  for (let i = 0; i < arr.length; i++) {
    if (arr[i] % 2 == 0) {
      filteredArr.push(arr[i]);
    }
  }
  return filteredArr;
}
console.log(filterEven(arr));

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

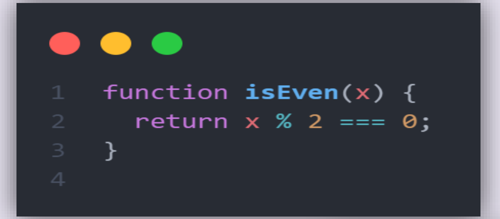


Examples of Higher Order Functions

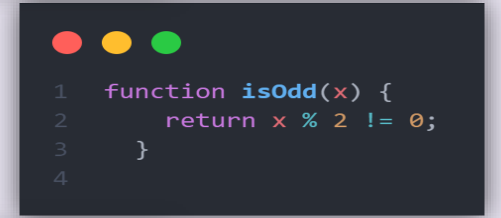
Q: Suppose this given array **arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]**



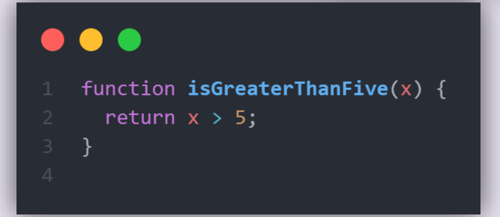
```
function filterFunction(arr, callback) {
    const filteredArr = [];
    for (let i = 0; i < arr.length; i++) {
      callback(arr[i]) ? filteredArr.push(arr[i]) : null;
    }
    return filteredArr;
  }
```



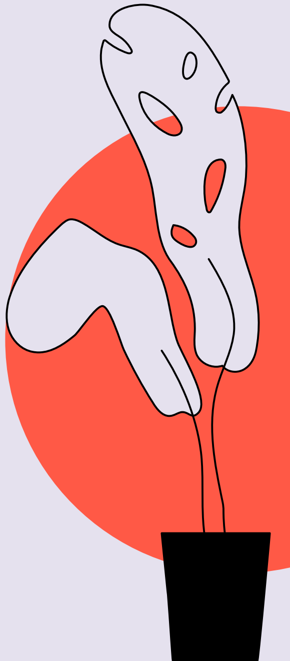
```
function isEven(x) {
  return x % 2 === 0;
}
```



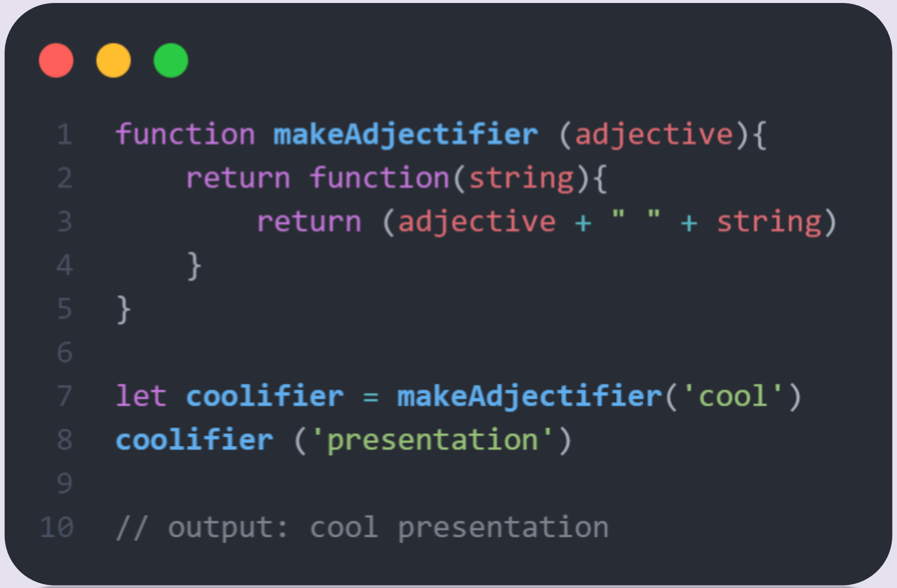
```
function isOdd(x) {
    return x % 2 != 0;
  }
```



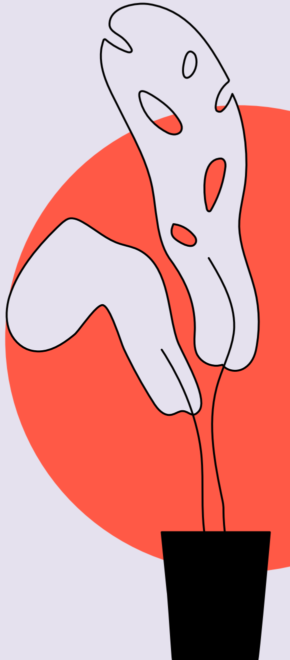
```
function isGreaterThanFive(x) {
  return x > 5;
}
```



Examples of Higher Order Functions

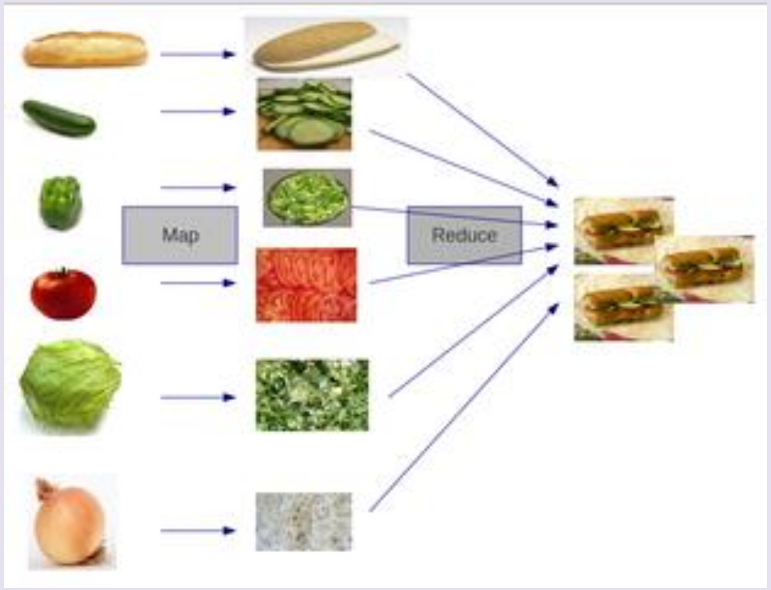


```
function makeAdjectifier (adjective){
    return function(string){
        return (adjective + " " + string)
    }
}

let coolifier = makeAdjectifier('cool')
coolifier ('presentation')

// output: cool presentation
```



05 Don’t Iterate

Don’t use Loops …. Use Array methods



06 Avoid Mutability

- Mutation

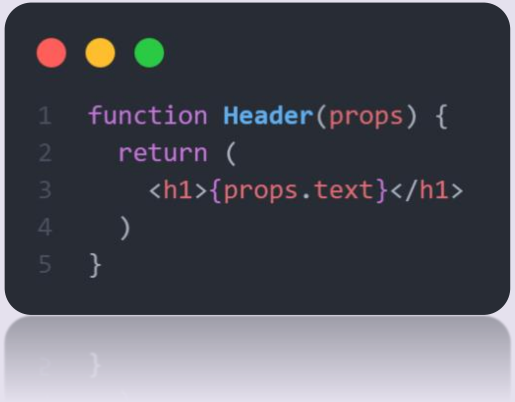
```
var rooms = [“H1”, “H2”, “H3”];
rooms[2] = “H4”;
rooms;
=> ["H1", "H2", "H4"]
```

- No Mutation

```
var rooms = [“H1”, “H2”, “H3”];
Var newRooms = rooms.map(function (rm) {
if (rm == “H3”) { return “H4”; }
else { return rm; }
});
```

07 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 ?