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.

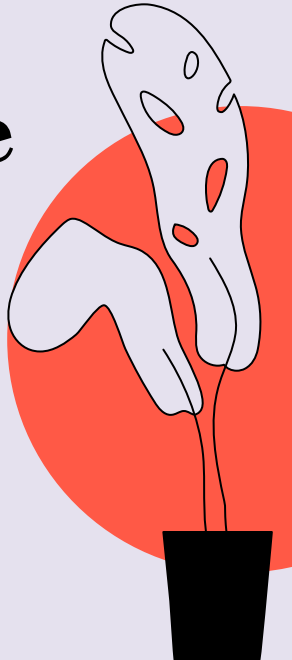
02 Functional Programming

- **Imperative Programming**
- **Declarative Programming**

Imperative VS Declarative

Imperative

Declarative



Imperative VS Declarative

- Imperative Programming

programming style that we specify the program logic, by describing the flow control

```
let imperative = "HOW?"
```

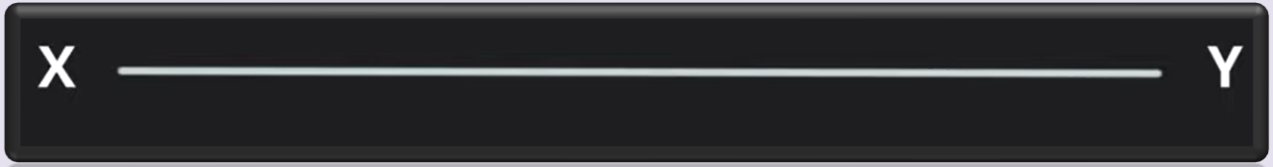
Imperative VS Declarative

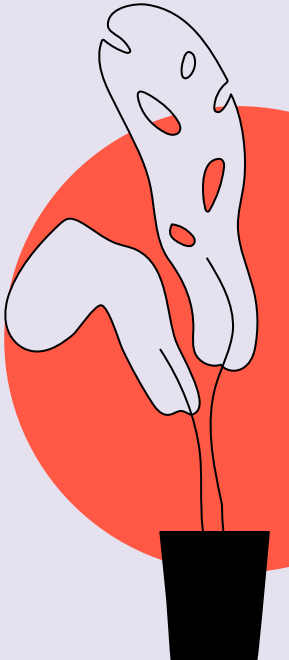
- Declarative Programming

programming style that we specify the program logic, without describing the flow control

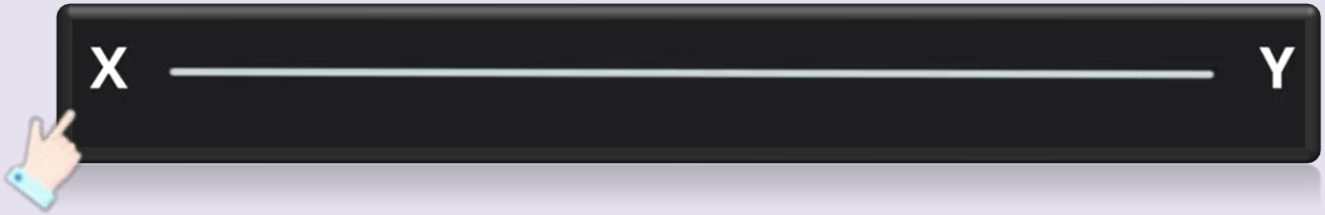
```
let declarative = "WHAT?"
```

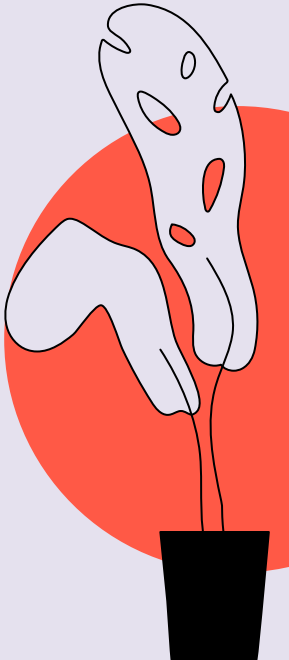
Imperative VS Declarative



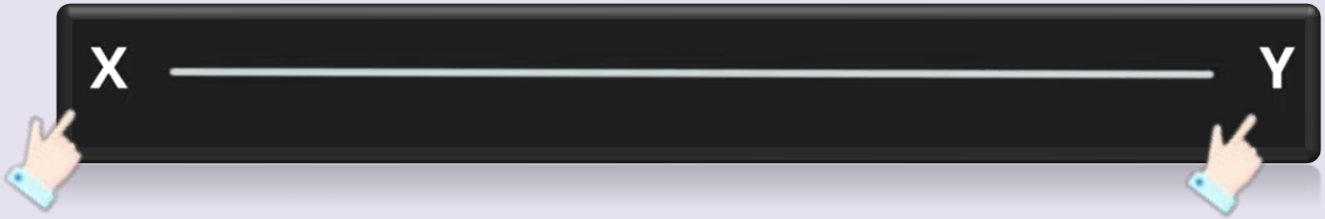


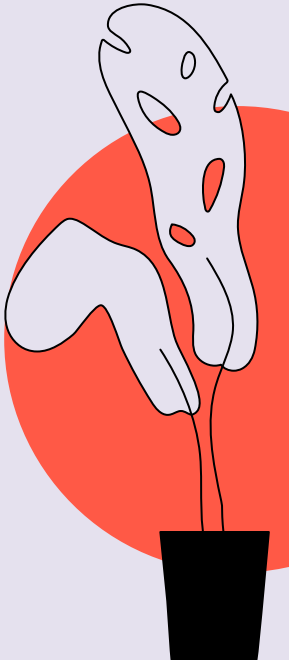
Imperative VS Declarative



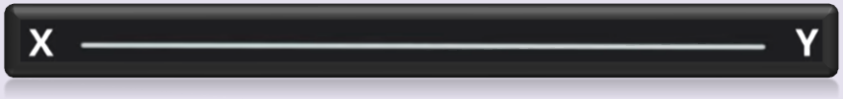


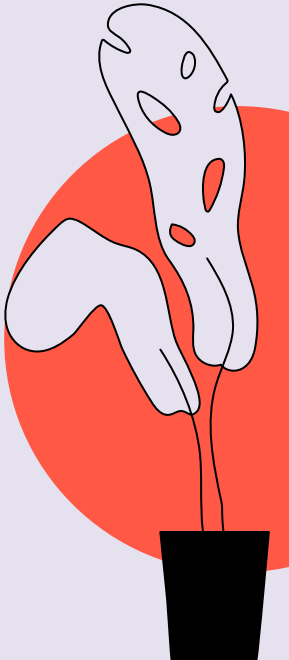
Imperative VS Declarative



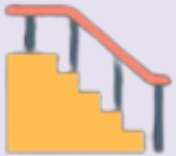


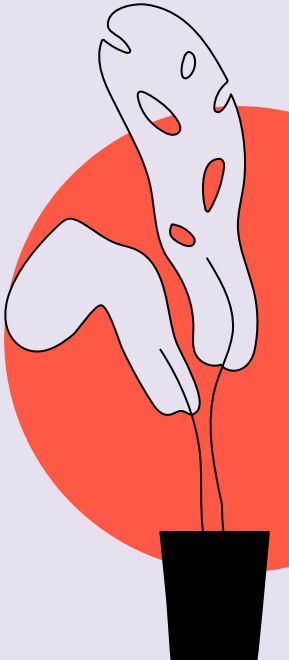
Imperative



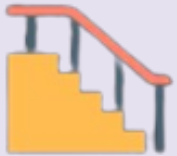


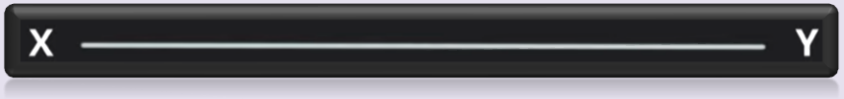
Imperative

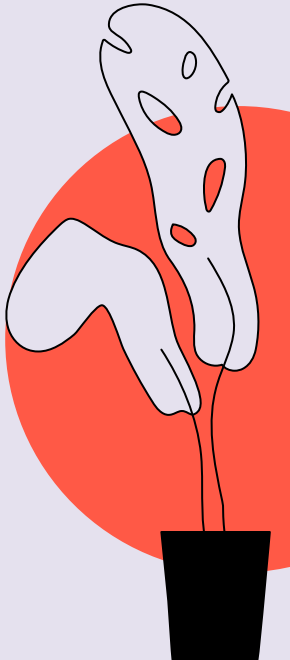




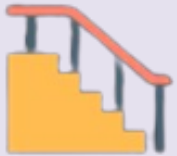
Imperative

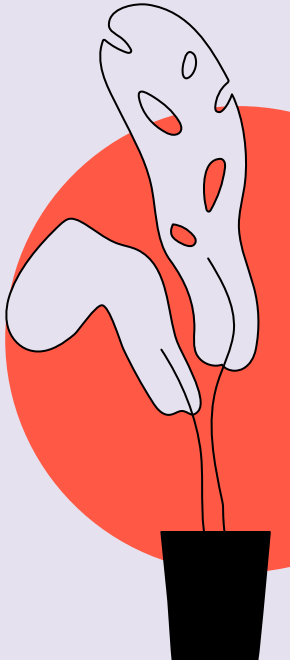




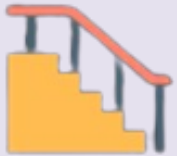


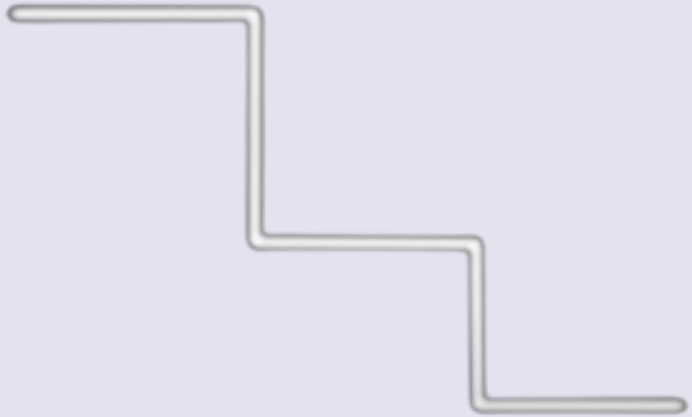
Imperative

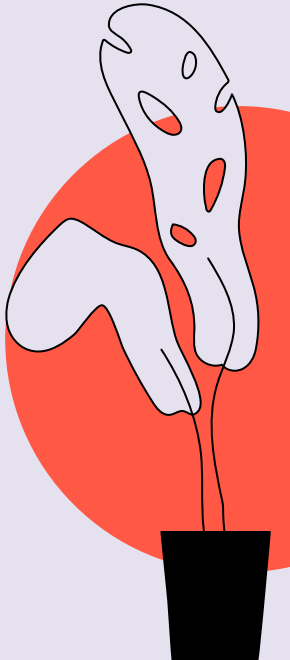




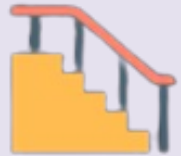
Imperative

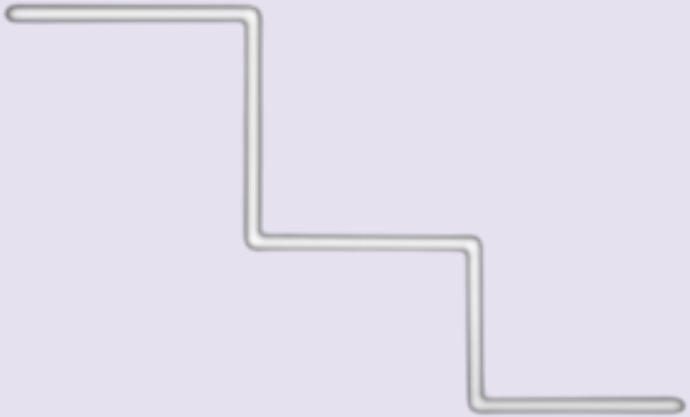


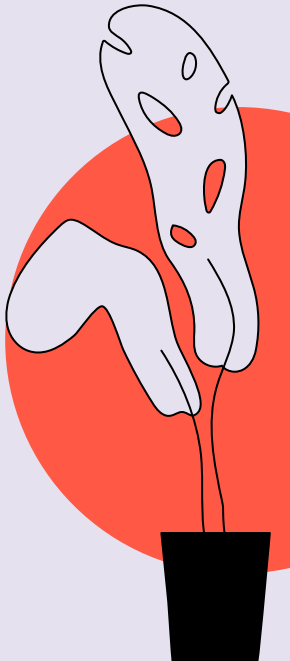




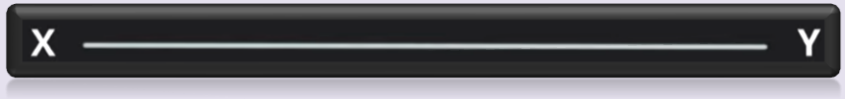
Imperative

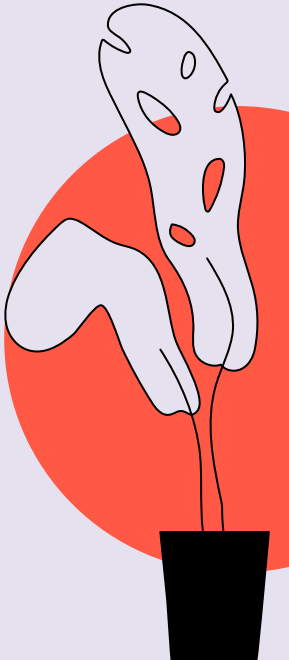




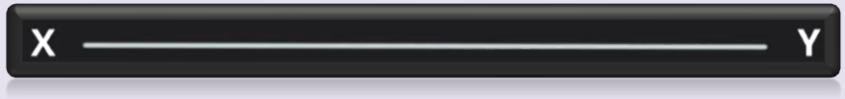


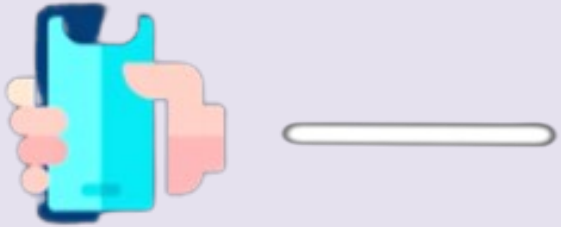
Declarative

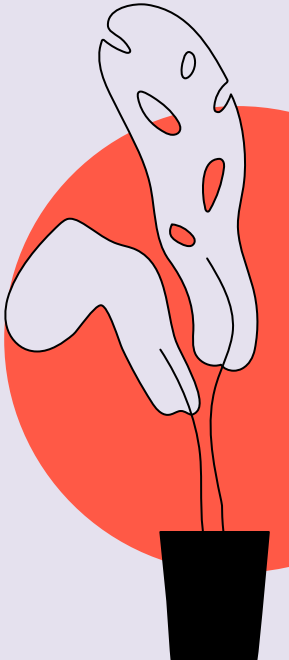




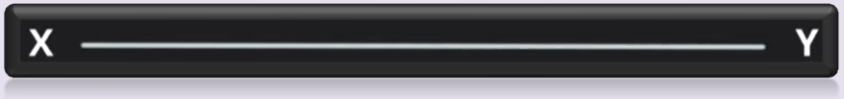
Declarative

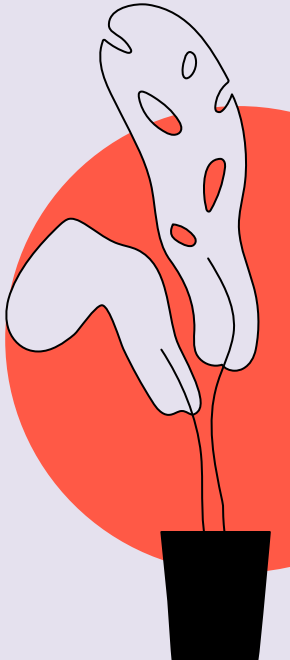




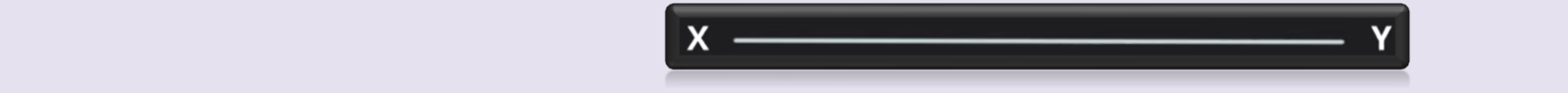


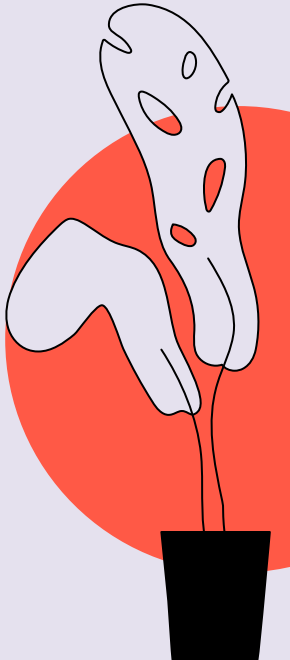
Declarative



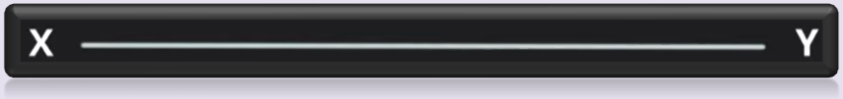


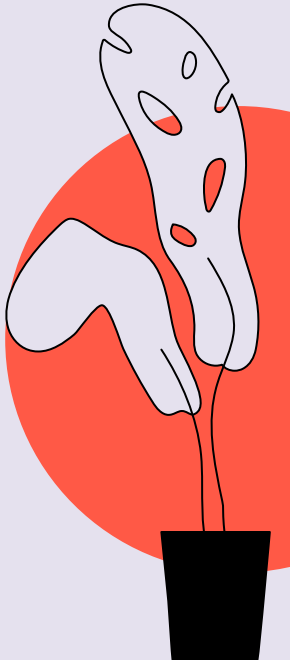
Declarative





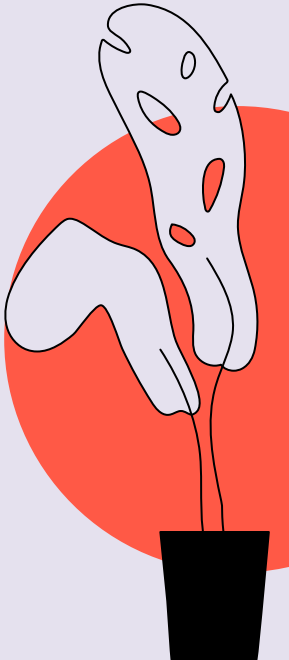
Declarative





Examples of Imperative programming :

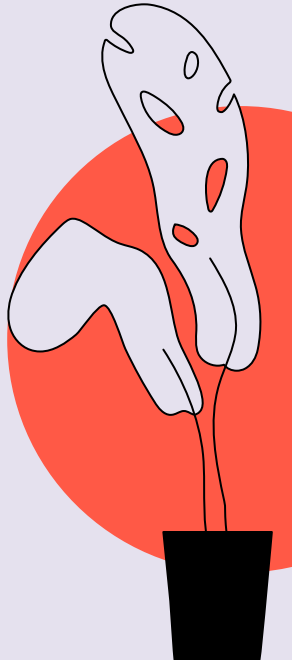
```
function getSum(arr) {
  let sum = 0;
  for (let i =0 ; i<arr.length; i++) {
    sum+=arr[i];
  }
  return sum;
}
```



Examples of Imperative programming :

```
function getSum(arr) {
  let sum = 0;
  for (let i =0 ; i<arr.length; i++) {
    sum+=arr[i];
  }
  return sum;
}
```

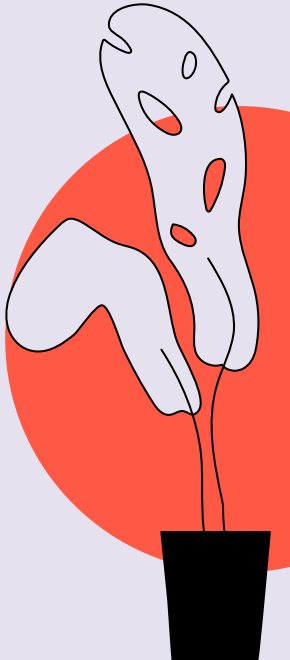
- Declare variable



Examples of Imperative programming :

```
function getSum(arr) {
  let sum = 0;
  for (let i =0 ; i<arr.length; i++) {
    sum+=arr[i];
  }
  return sum;
}
```

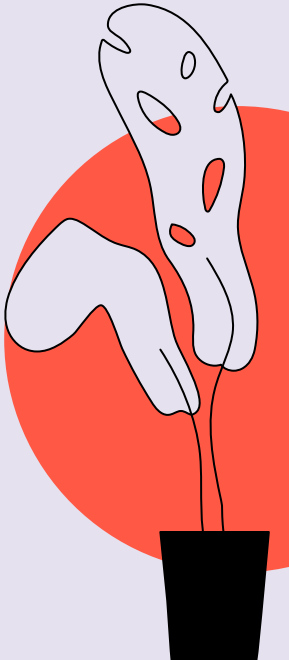
- Declare variable
- Loop over array



Examples of Imperative programming :

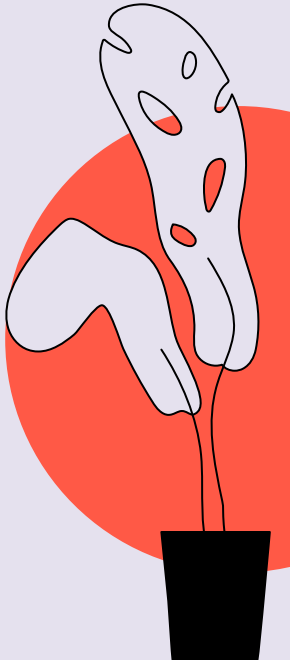
```
function getSum(arr) {
  let sum = 0;
  for (let i =0 ; i<arr.length; i++) {
    sum+=arr[i];
  }
  return sum;
}
```

- Declare variable
- Loop over array
- Return result



Examples of Imperative programming :

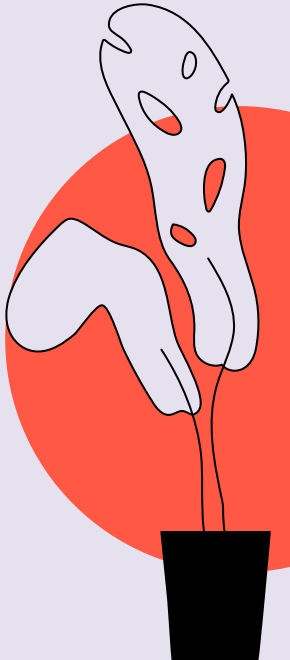
```
function getMax(arr) {
  let max = -Infinity;
  for (let i = 0; i < arr.length; i++) {
    max = arr[i] > max ? arr[i] : max;
  }
  return max;
}
```



Examples of Imperative programming :

```
function getMax(arr) {
  let max = -Infinity;
  for (let i = 0; i < arr.length; i++) {
    max = arr[i] > max ? arr[i] : max;
  }
  return max;
}
```

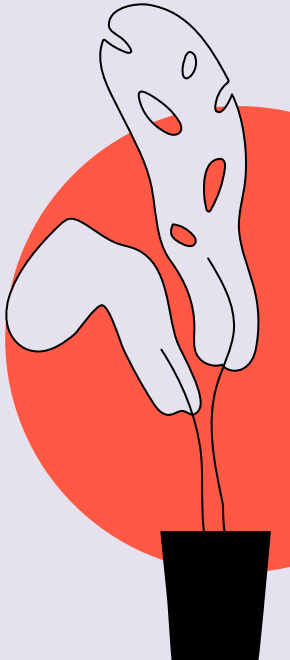
- Declare variable



Examples of Imperative programming :

```
function getMax(arr) {
  let max = -Infinity;
  for (let i = 0; i < arr.length; i++) {
    max = arr[i] > max ? arr[i] : max;
  }
  return max;
}
```

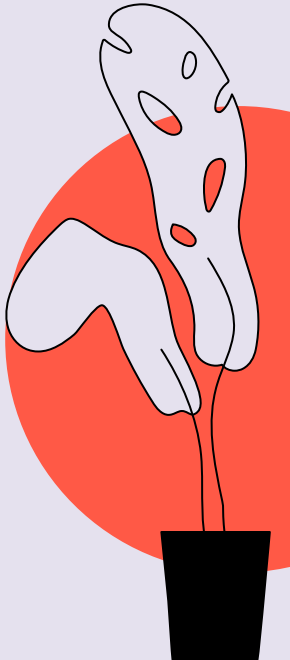
- Declare variable
- Loop over array



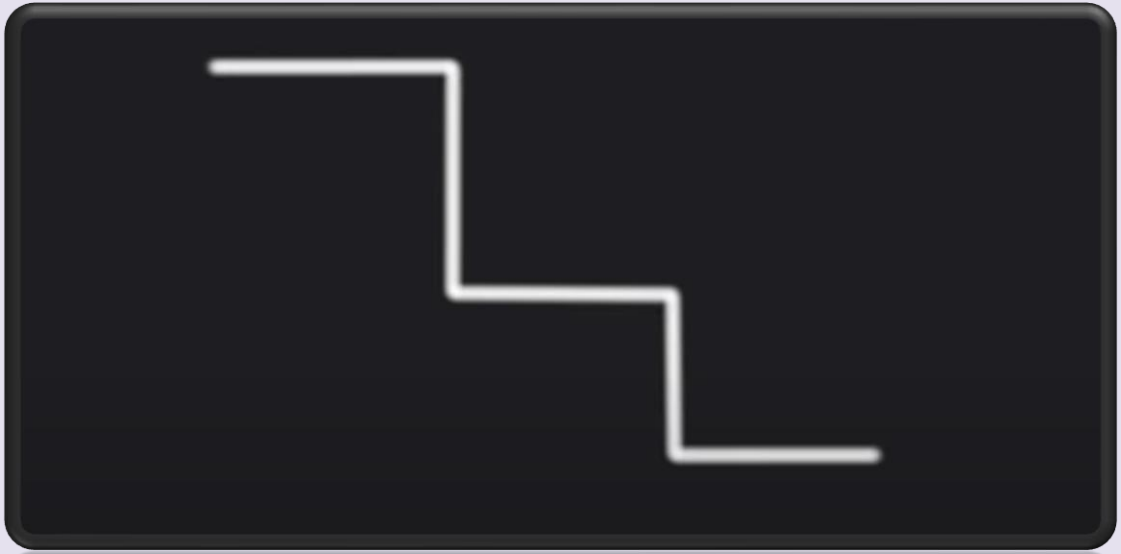
Examples of Imperative programming :

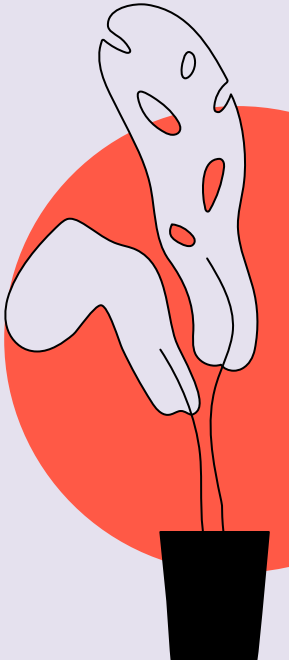
```
function getMax(arr) {
  let max = -Infinity;
  for (let i = 0; i < arr.length; i++) {
    max = arr[i] > max ? arr[i] : max;
  }
  return max;
}
```

- Declare variable
- Loop over array
- Return result

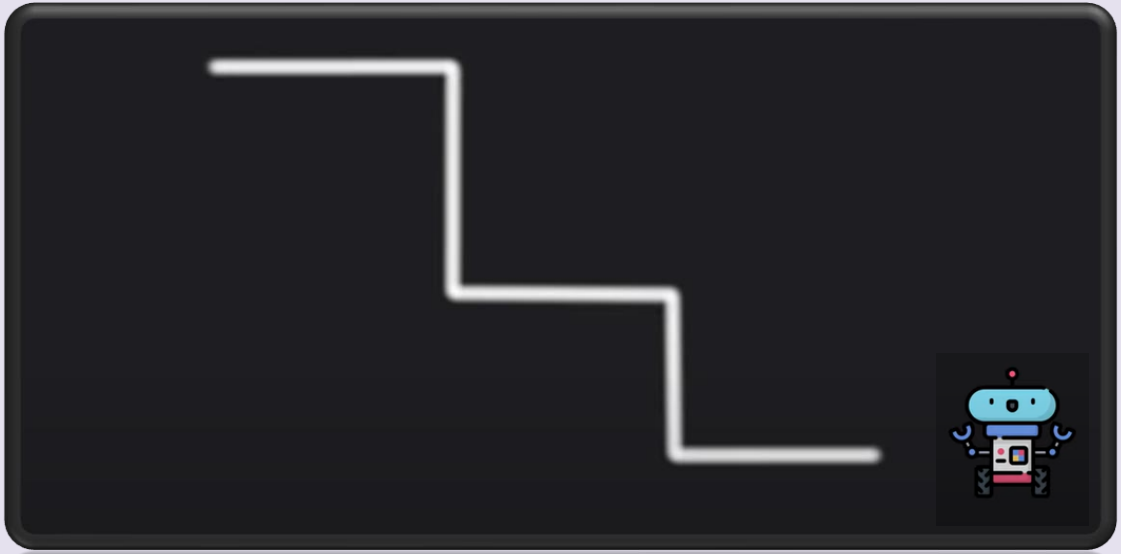


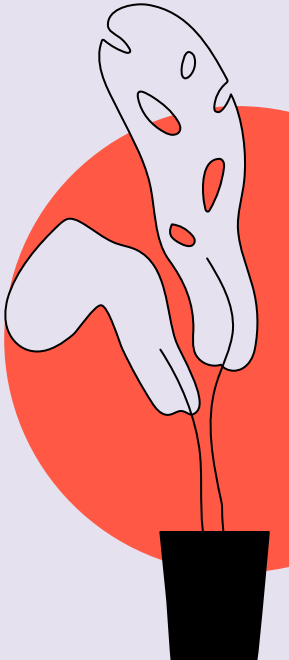
Imperative Programming





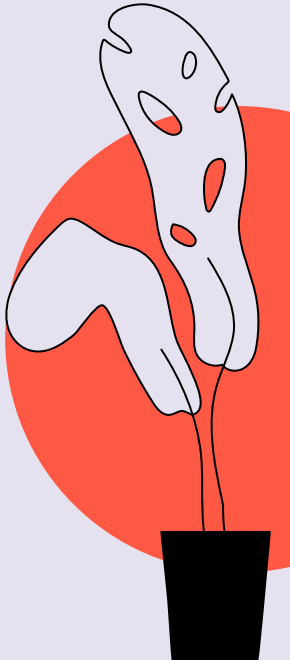
Imperative Programming





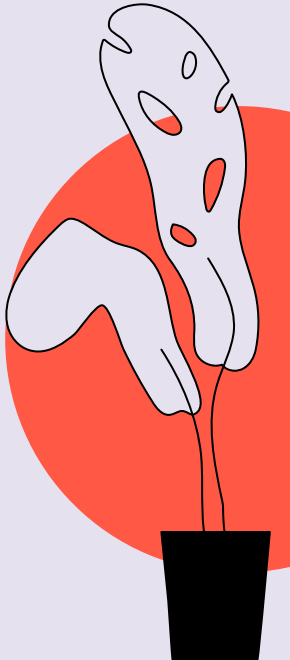
Examples of Declarative programming :

```
function getSum(arr) {
  return arr.reduce((prev, curr) => prev + curr, 0);
}
```



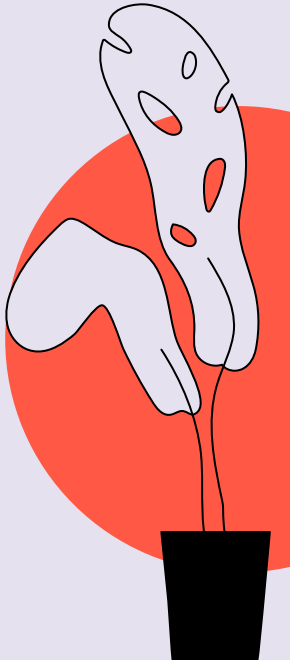
Examples of Declarative programming :

```
function getSum(arr) {
  return arr.reduce((prev, curr) => prev + curr, 0);
}
```



Examples of Declarative programming :

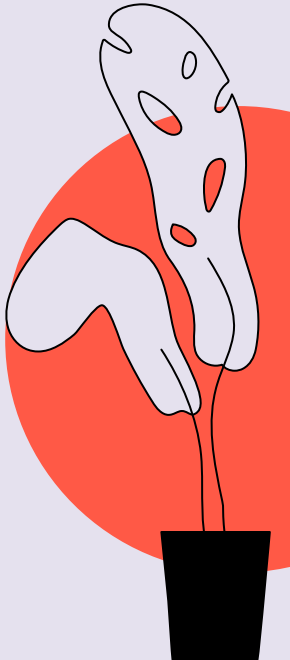
```
function getSum(arr) {
  return arr.reduce((prev, curr) => prev + curr, 0);
}
```



Examples of Declarative programming :

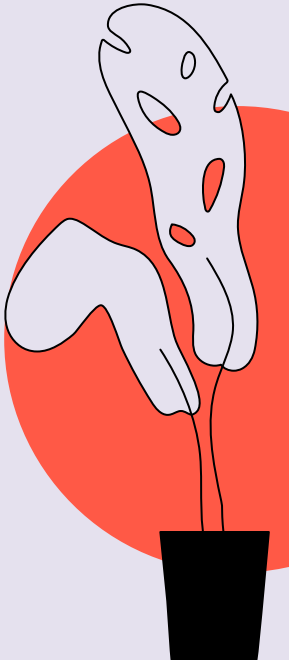
```
function getSum(arr) {
  return arr.reduce((prev, curr) => prev + curr, 0);
}
```

```
What do we want?
The sum of the array elements.
```



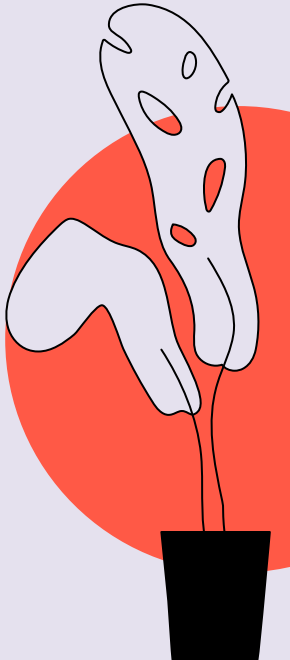
Examples of Declarative programming :

```
function getMax(arr) {
  return Math.max(...arr);
}
```



Examples of Declarative programming :

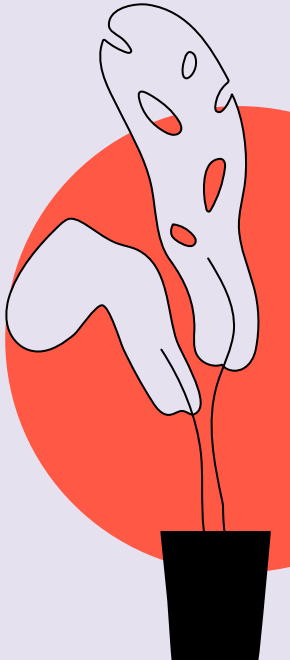
```
function getMax(arr) {
  return Math.max(...arr);
}
```



Examples of Declarative programming :

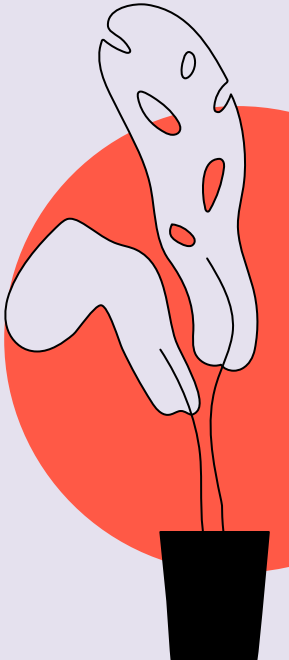
```
function getMax(arr) {
  return Math.max(...arr);
}
```

```
What do we want?
The maximum element in the array
```



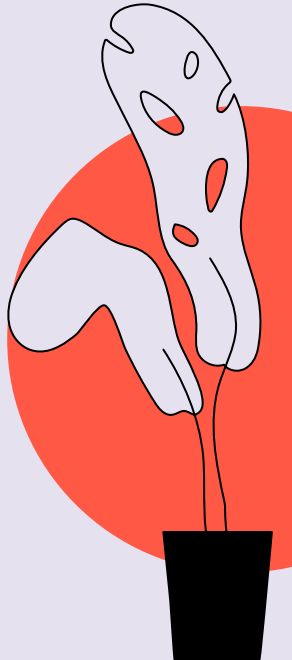
Examples of Declarative programming :

```
SELECT * FROM students WHERE age > 15
```

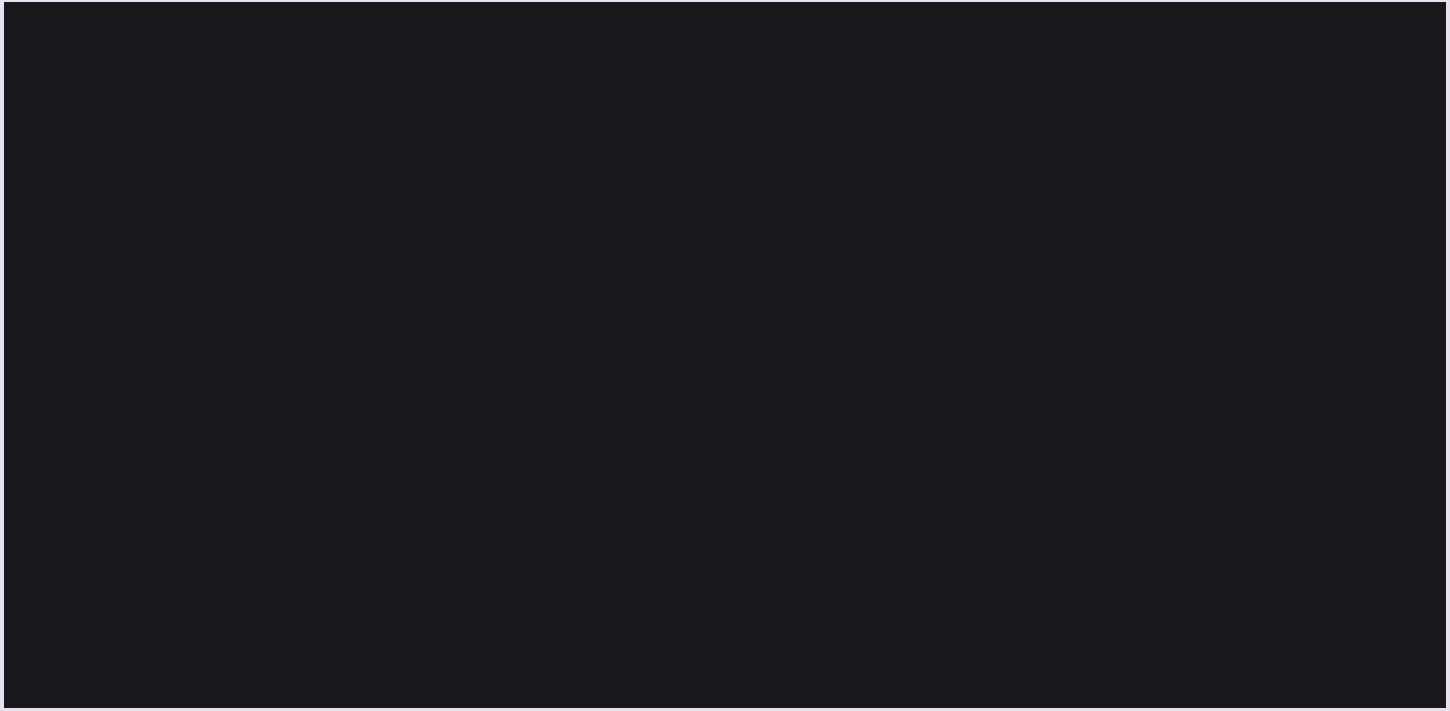


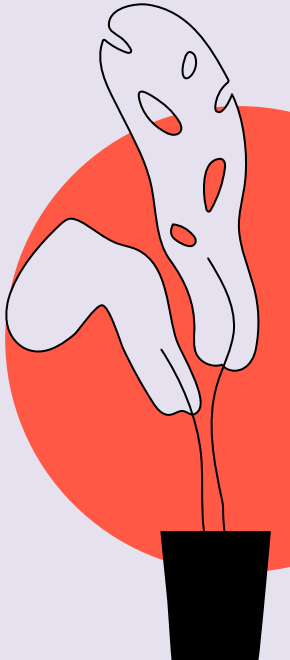
Examples of Declarative programming :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Hello World!</h1>
</body>
</html>
```



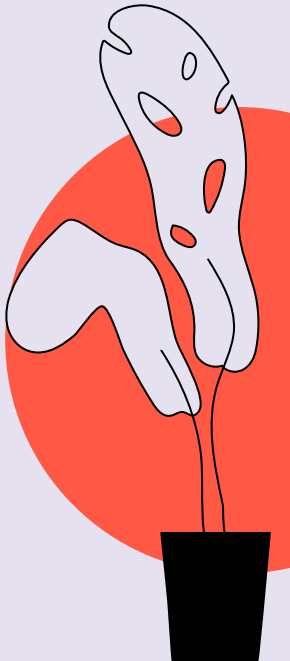
Imperative VS Declarative



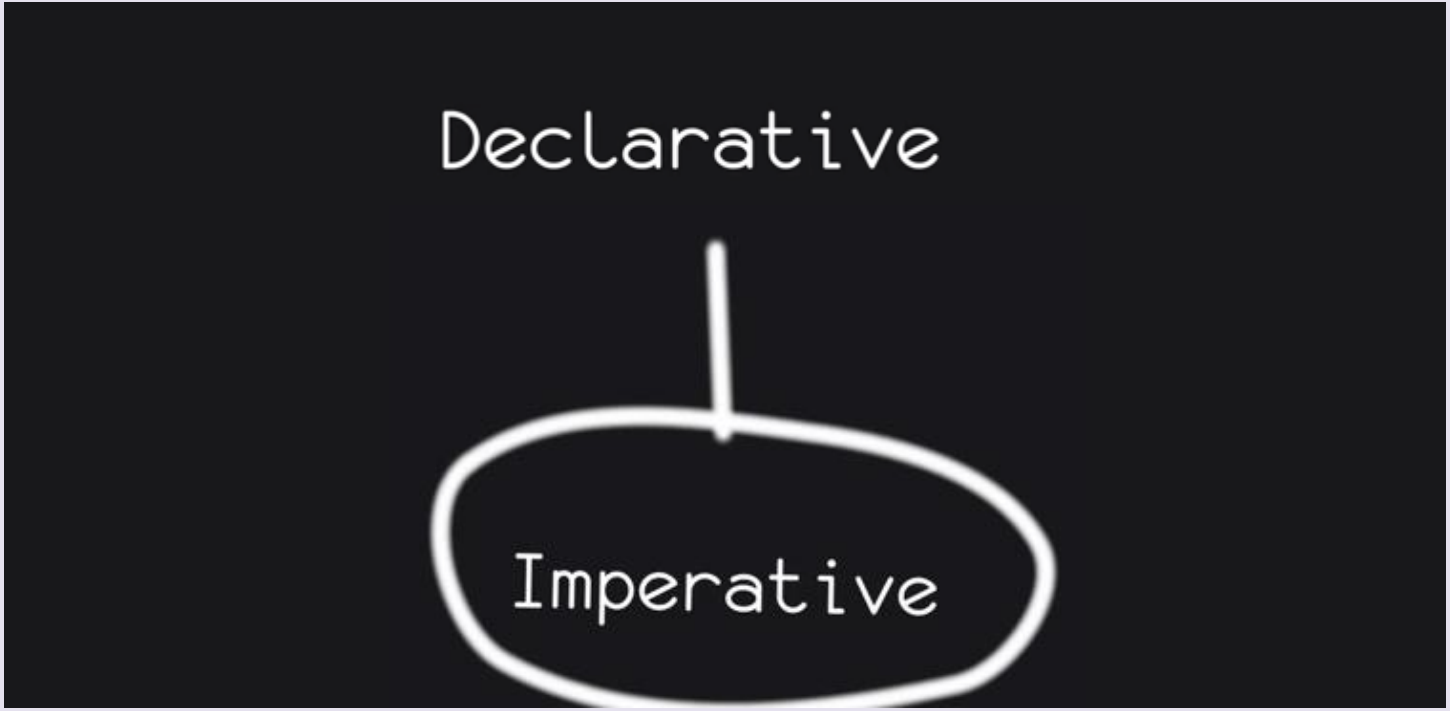


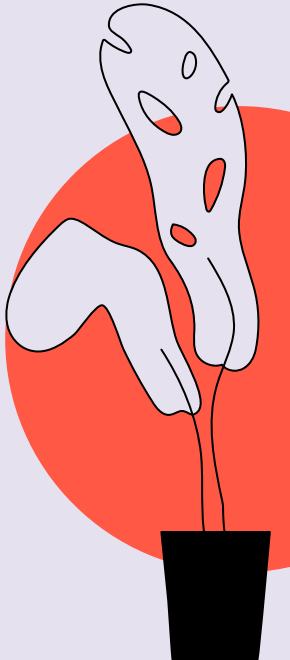
Imperative VS Declarative

Declarative

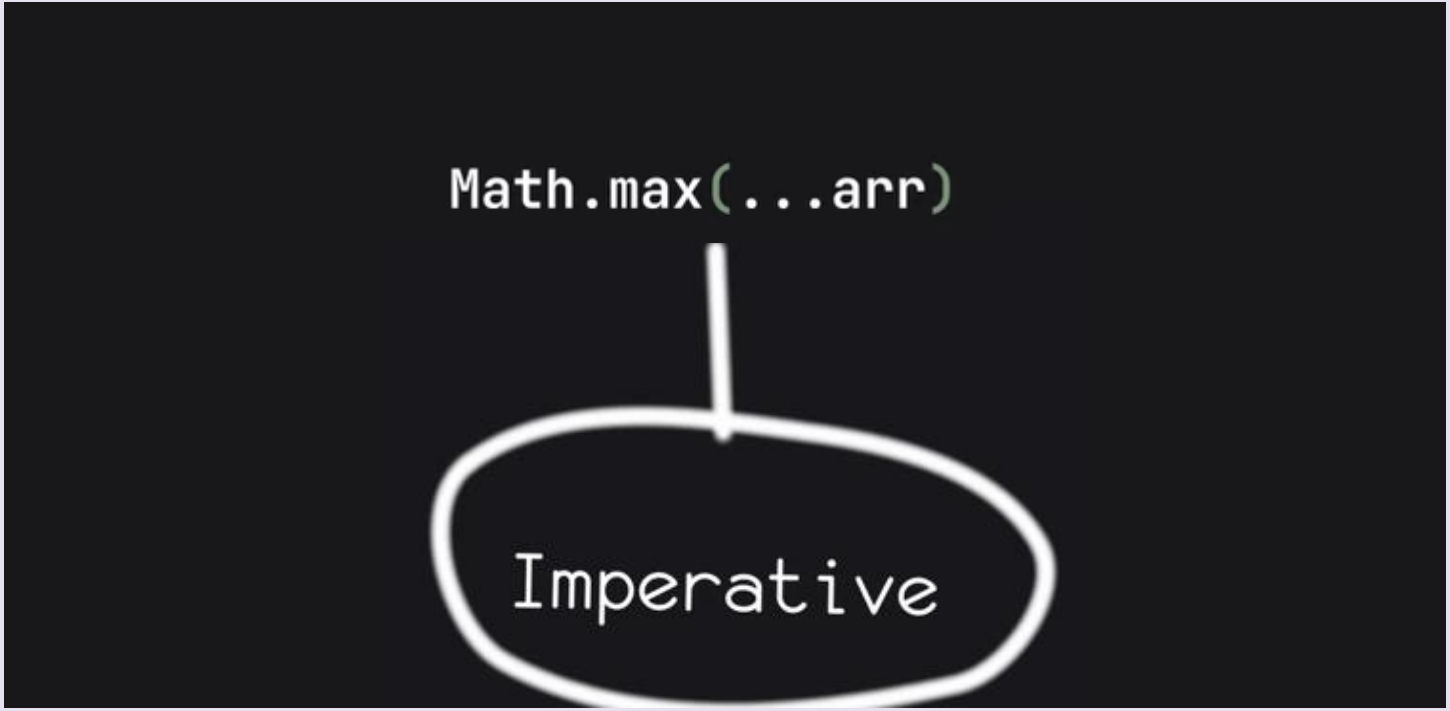


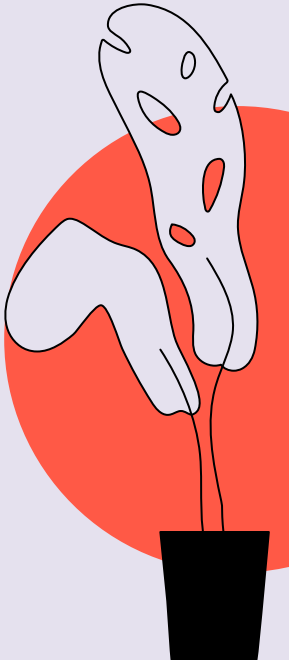
Imperative VS Declarative



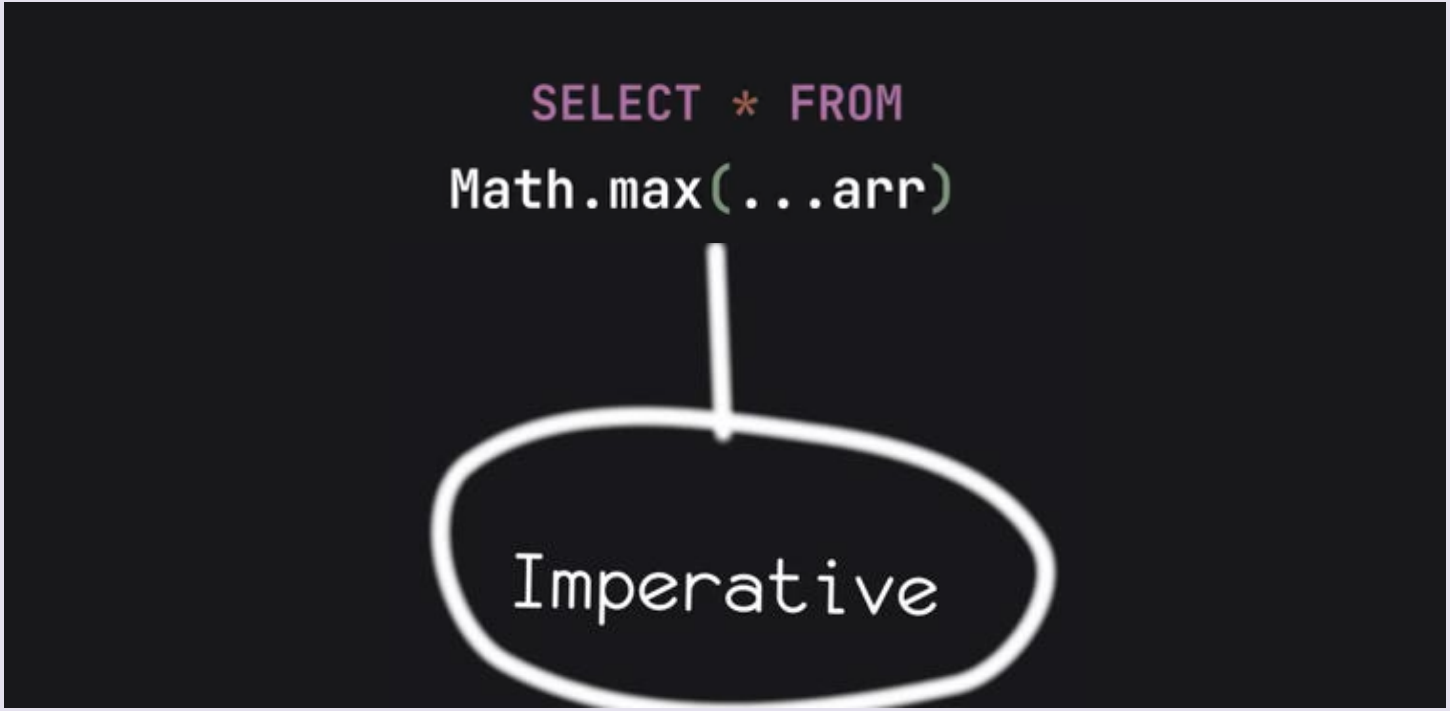


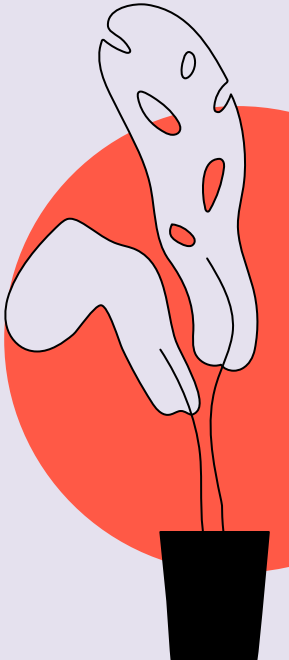
Imperative VS Declarative





Imperative VS Declarative

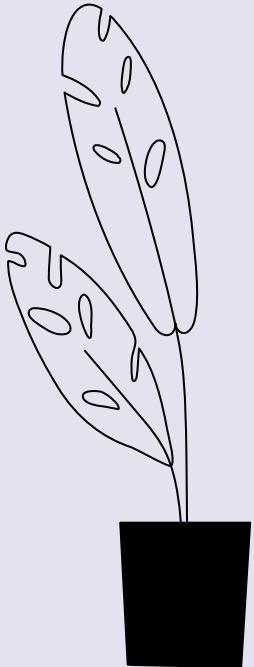




03 Pure Functions

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.



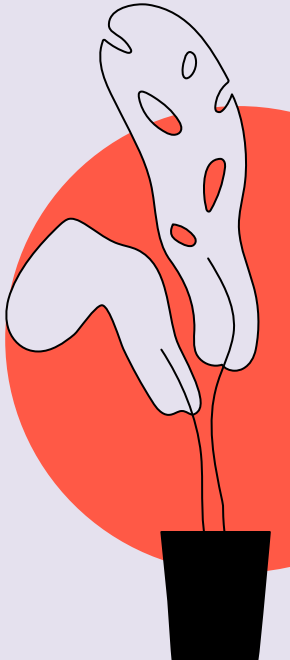
Examples of Pure and Not Pure Functions

```
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

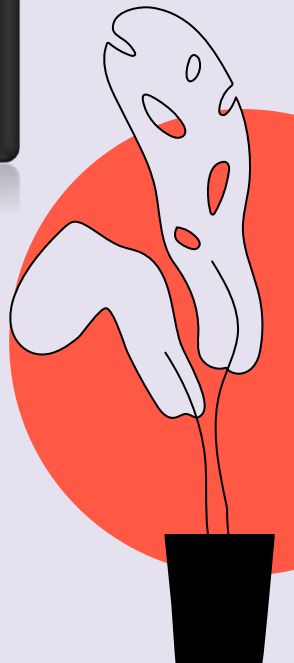
Q: Suppose this given array **arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]**

```
function 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;
}
```

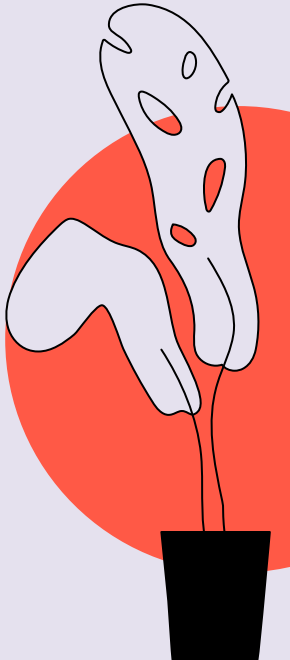


Examples of Higher Order Functions

```
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 ?