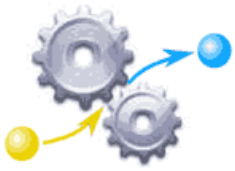
functional programming

What is a Function?

📌 A function relates an input to an output.



It is like a machine that has an input and an output.

And the output is related somehow to the input.

f(x)	"**f(x) = ...** " is the classic way of writing a function. And there are other ways, as you will see!

📌 function is an object in javascript. i.e function is first-class-citizen like values and objects

---

how create function ?

2 ways

1. function declaration / named function
2. function expression / anonymous function

📌 go live 👨‍💻

---

function declaration / named function

```
function add(n1, n2) {
    let result = n1 + n2
    return result
}
```

---

function expression / anonymous function

```
let add = function (n1, n2) {
    return n1 + n2
```

```
}
```

---

function parameters

📌 function can take zero or more params,not restricted by function argument names

```
function f1(a, b) {
    console.log(arguments[0])
    console.log(arguments[1])
    console.log(arguments[2])
    console.log(arguments[3])
}

f1(10,20,30,40)
f1()
```

```
function sum() {
    let len = arguments.length,
        result = 0,
        i = 0;
    while (i < len) {
        result += arguments[i]
        i++
    }
    return result
}
```

▼ can we do function overloading by params ?

No

```
function getFood() {
    return "No Food"
}
function getFood(pay) {
    //..
    return "Food"
}
function getLunch(pay, extra) {
    //..
    return "Food" + "  snacks"
}
```

One Naive Solution:

```
function getLunch() {
    if (arguments.length === 0)
        return "No Lunch"
    if (arguments.length === 1)
        return "Biryani"
    if (arguments.length === 2)
        return "biryani" + "  soft-drink"
}
```

---

function with default & rest parameters

```
function f(a = 1, b = 2, ...c) {

    // if (!a)
    //     a = 1
    // if (!b)
    //     b = 2

    // or

    // a = a || 1
    // b = b || 2

    console.log(a)
```

```
    console.log(b)
    console.log(c[1])

}
f(10, 20, 30, 40, 50, 60, 70)
```

Quiz

```
function f(...r, x, y) {
    console.log(r)
    console.log(x)
    console.log(y)
}
f(10,20,30)  // Syntax Error
```

Functional Programming Principles

- A function can be stored in a variable

```
function greet() {
    console.log("hello")
}
let sayHello = greet

// sayHello()
```

- A parameter of a function can be a function

```
function greet(f) {
    console.log("🌹🌹🌹🌹🌹🌹🌹🌹🌹🌹")
    if (f)
        f()
    else
        console.log("welcome")
    console.log("🌹🌹🌹🌹🌹🌹🌹🌹🌹🌹")
}
// greet()

//-------------
// in india
//-------------
let tn_greet = function () {
    console.log("வரவேற்பு")
}
// greet(tn_greet)

//-------------
// in UAE
//-------------
let ar_greet = function () {
    console.log("أهلا بك")
}
// greet(ar_greet)
```

- The return value of a function can be a function

```
function teach() {
    console.log("teaching javascript....")
    //..
    let learn = function () {
        console.log("learning javascript...")
    }
    //learn();
    console.log("teaching ends...")
    return learn;
}

let learnFn = teach();

learnFn()
learnFn()
```

Higher Order Function / Higher Order Programming

> 📌 In mathematics and computer science, a higher-order function is a function that does at least one of the following: **takes one or more functions as arguments, returns a function as its result. All other functions are first-order functions**.

code without hof

```
function hello(){
     console.log("hello..")
     console.log('😀')
 }
 function hi(){
     console.log("hi")
     console.log("😀")
 }

 hello()
 hi()
```

design issues:

- code duplication
- code tight-coupling

solution:

📌 use **Higher Order Function / Higher Order Programming**

---

code with hof

```
// e.g HOF
function withEmoji(f) {
    return function () {
        f()
        console.log('😀')
    }
}

function hello() {
    console.log("hello")
}
function hi() {
    console.log("hi")
}

let helloWithEmoji = withEmoji(hello);
// hello()
// helloWithEmoji()
```

Function Closure

📌 A closure is a function having access to the parent scope,even after the parent function has closed.

Ex.

```
function teach(sub) {
    console.log("teaching " + sub)
    let notes = sub + "-notes"
    let fun = "🤣🤣🤣🤣🤣🤣🤣"
    let learn = function () {
        console.log("learning with " + notes)
    }
    //learn()
    console.log("teaching ends")
    return learn
}

let learnFn = teach("javascript");
learnFn()
learnFn()
learnFn()
```

---

when / where we need closure ?

📌 to abstract public behavior of javascript module

Ex. e.g counter module

```
const counter = (function () {
    console.log("init()")
    let count = 0  // private
    // public
    function increment() {
        count++
    }
    function get() {
        return count
    }
    return {
        inc: increment,
        get: get
```

```
    }
})();
```

self-executable function / IIFE (immediately invokable function expression)

📌 global scope is bad in javascript

🧿 style of coding