



## Hof (higher Order function)

↳ these are those functions which accept, another func<sup>n</sup> as an argument & use them.

## Callbacks

↳ these are those func<sup>n</sup> which are passed as an argument to a higher order func<sup>n</sup>

forEach ✓ Hof

Math.sqrt

→ X hof

```
arr.map(function f() {});
```

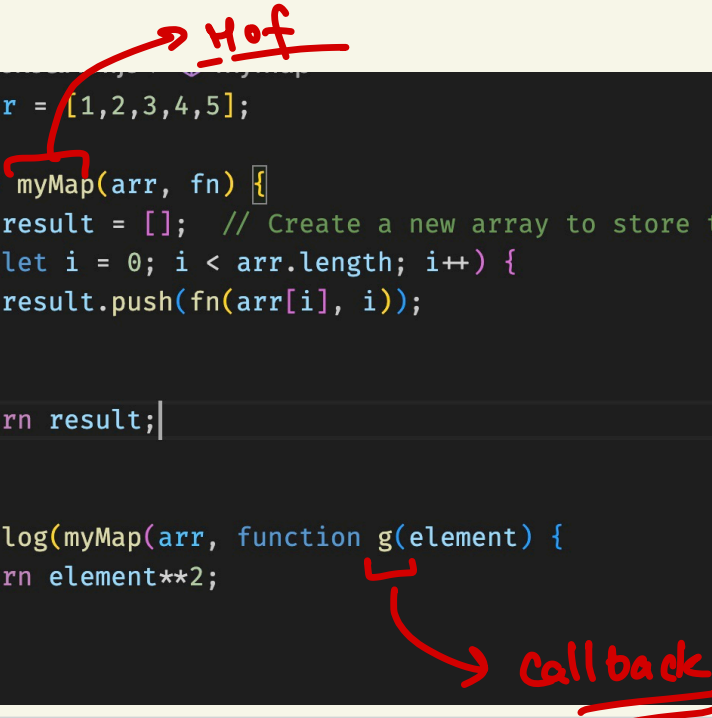
hof → why?!

Because it is accepting  
func<sup>n</sup> "f" as an argument

"f" is a callback

why? because "f" is  
being passed to  
map as an argument

```
1  const arr = [1,2,3,4,5];
2
3  function myMap(arr, fn) {
4      let result = []; // Create a new array to store the results
5      for(let i = 0; i < arr.length; i++) {
6          result.push(fn(arr[i], i));
7      }
8
9      return result;
10 }
11
12 console.log(myMap(arr, function g(element) {
13     return element**2;
14 })))
15
```



We can decide on the runtime what to send as an implementation of callback.

```
if ( a < b )  
    myMap (arr, g);  
else if ( b < d )  
    myMap (arr, h);  
else  
    myMap (arr, k);
```

~~for g()~~

for h()

for k()

```

1  const arr = [1,2,3,4,5];
2
3  function myMap(arr, fn) {
4      let result = []; // Create a new array to store the results
5      for(let i = 0; i < arr.length; i++) {
6          result.push(fn(arr[i], i));
7      }
8
9      return result;
10 }
11
12 console.log(myMap(arr, function g(element) {
13     return element**2;
14 })))
15

```

$arr = [1, 2, 3]$

$[1, 4, 9]$

Call Stack

myMap  
 $result = [1, 4, 9]$   
 $i = 0, 1, 2, 3$   


---

myMap(arr,  
g);

$[1, 4, 9]$

## Disadvantages of a callback

- 1) callback hell (minor disadvantage, readability problem)
- 2) inversion of control

map (function  $f()$ )

4)

Trust

function  $f_1(f_n)$   $\hookrightarrow$  Hot

≡

}

function  $f_3(f_n)$   $\hookrightarrow$  Hot

≡

}

function  $f_2(f_n)$   $\hookrightarrow$  Hot

≡

}

function  $f_4(f_n)$   $\hookrightarrow$  Hot

≡

}



f1 (function () {

f2 (function () {

f3 (function () {

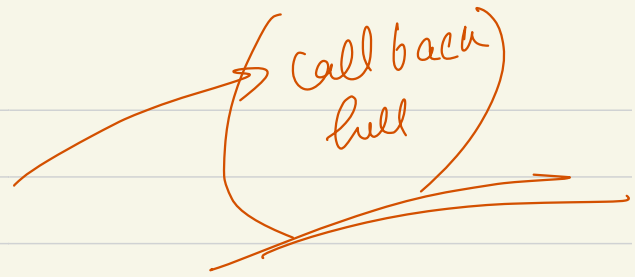
f4 (function () {

});

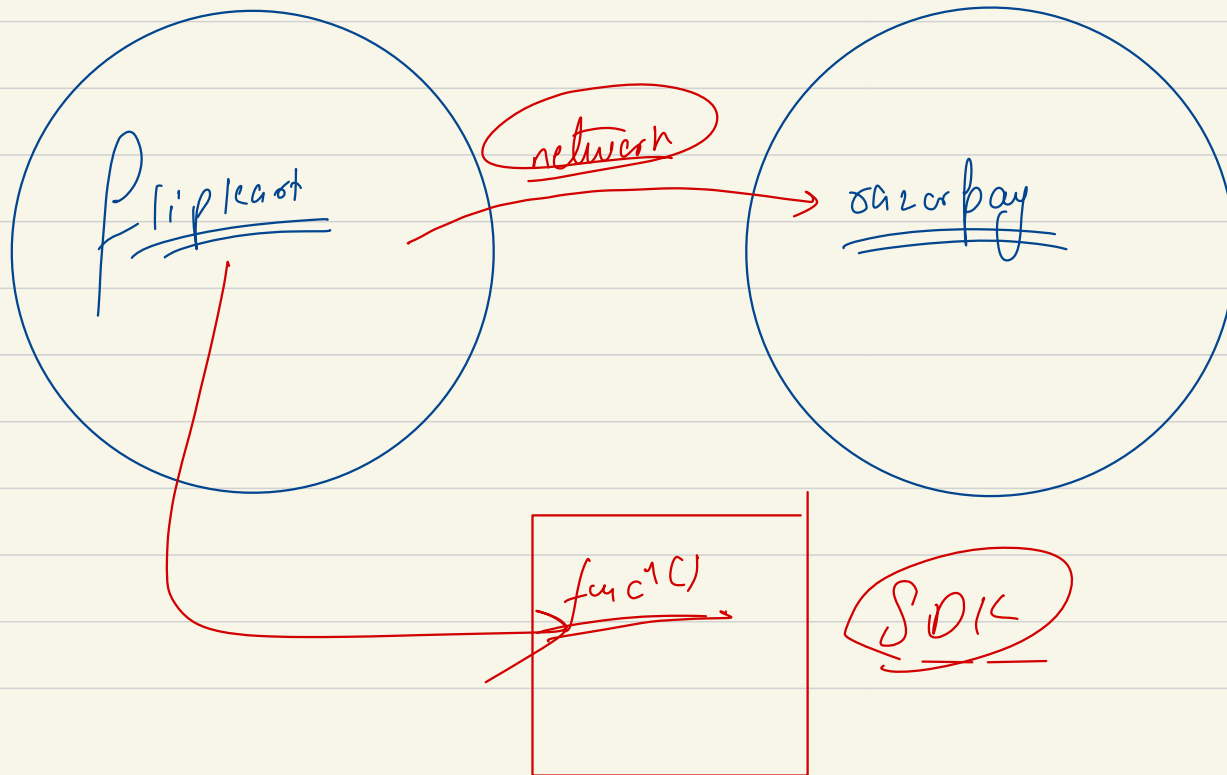
});

});

});



r



Trust

1000

Superlay

SDK

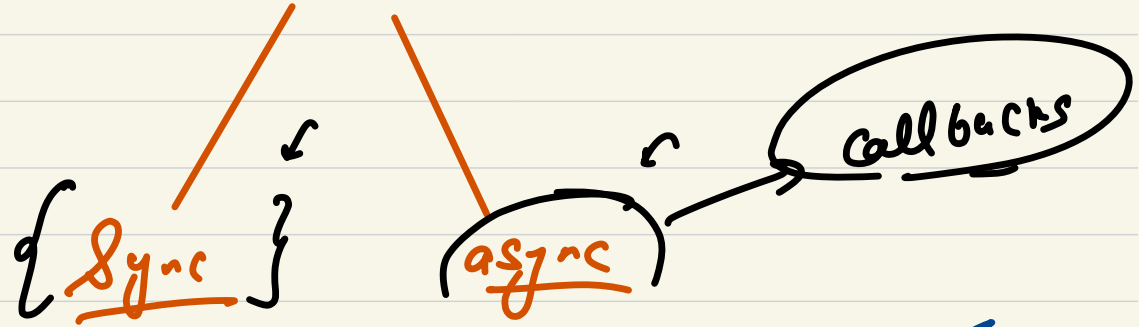
Superlay.makePayment

nightcode.io

(creds, function f() { } ) ;

2 £

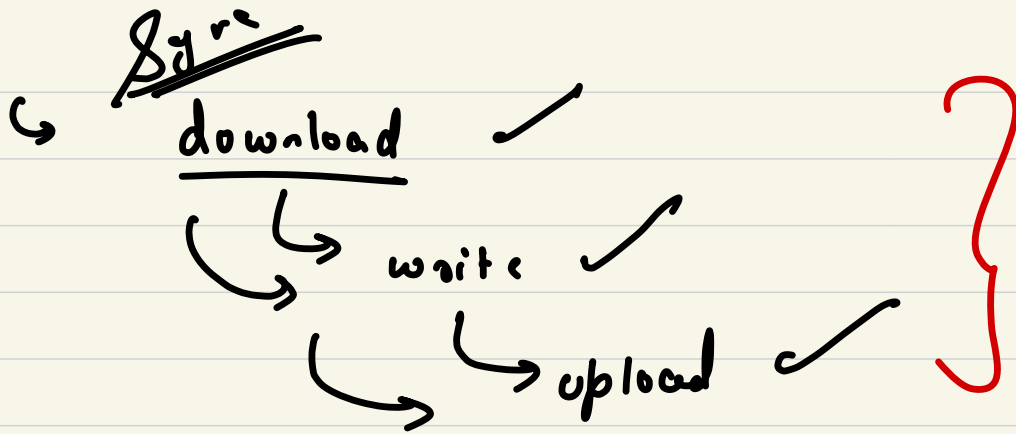
# Async Programming



Line 1 → ✓  
Line 2 → ✓  
Line 3 → ✓

Line 1  
Line 2  
Line 3

A diagram showing three lines of code, "Line 1", "Line 2", and "Line 3", written in red. To the left of each line is a blue arrow pointing downwards, indicating sequential execution. To the right of the lines are three blue curved arrows, each looping back to the line it just executed, representing a loop or a callback mechanism.



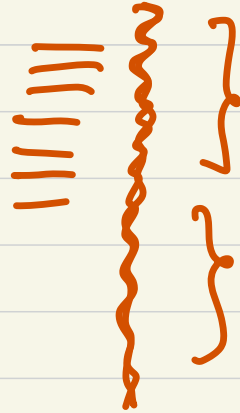
Case 2

- file 1 ✓
- file 2 ✓
- file 3 ✓
- file 4 ✓

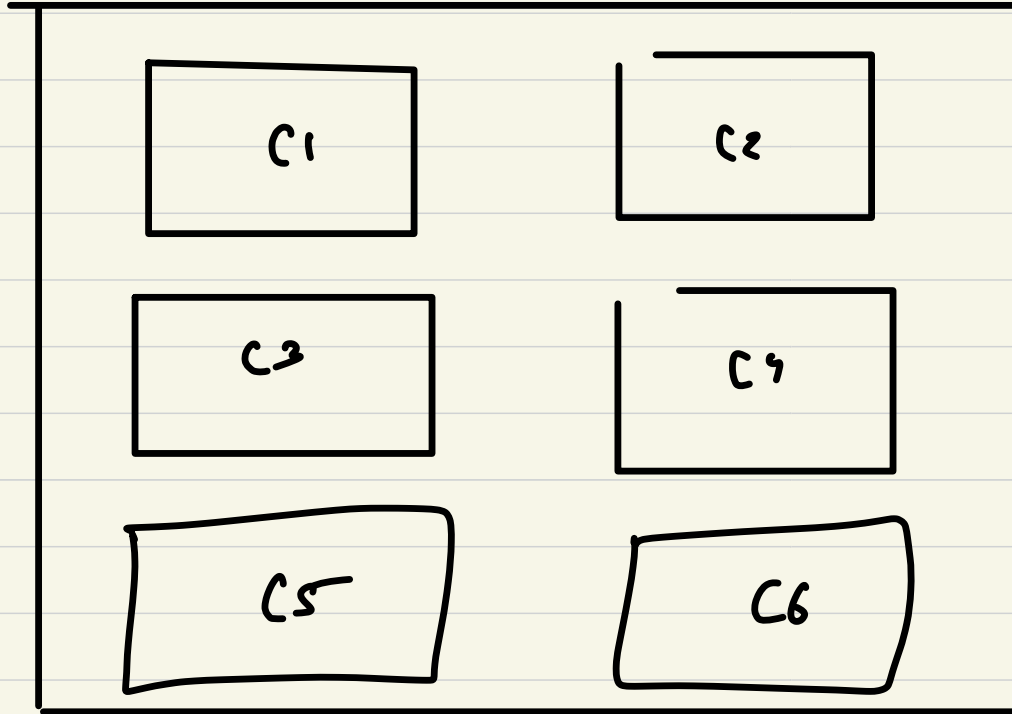
what is the default nature of JS ??

↳ any native piece of JS code is always  
sync in native.

↳ because JS is single threaded.



CPU



,

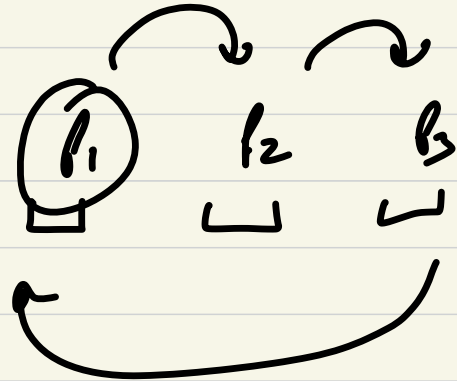
Context  
Switching

Single core machine

CPU  
1

X 2+ process parallel X  
X

1/sec  $\rightarrow$   $10^8 \sim 10^9$  instructions

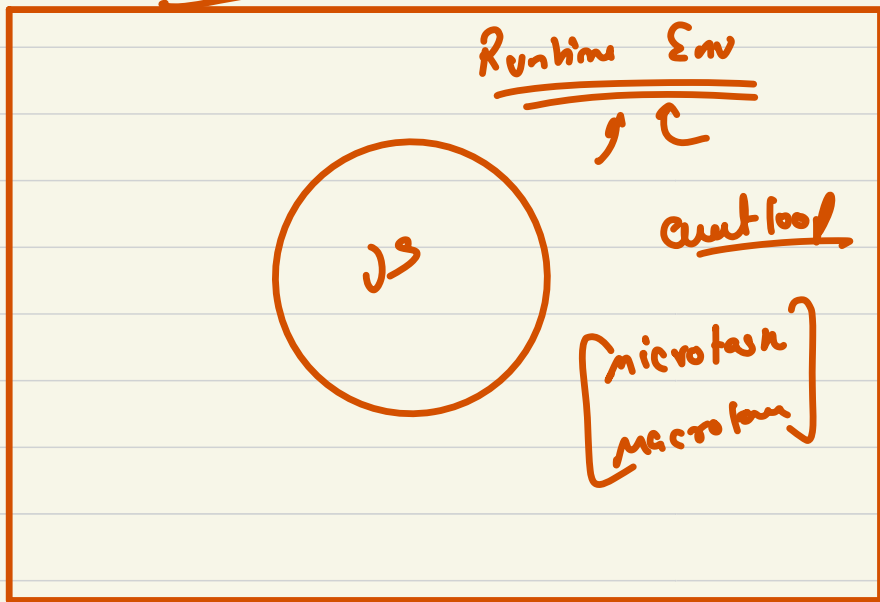


$p_1$     $p_2$     $p_3$   
Thread



Mozilla ✓

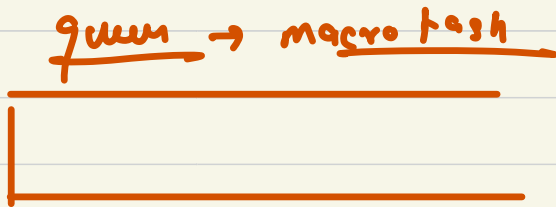
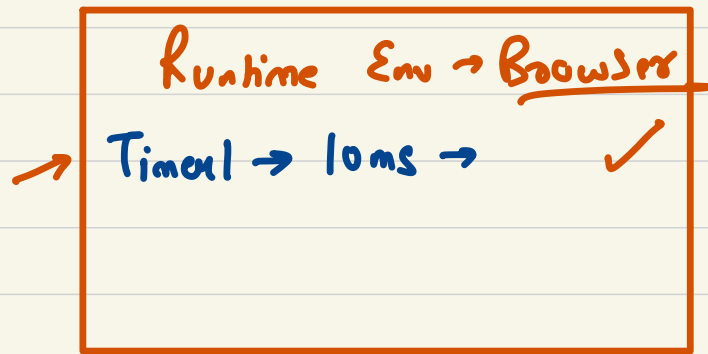
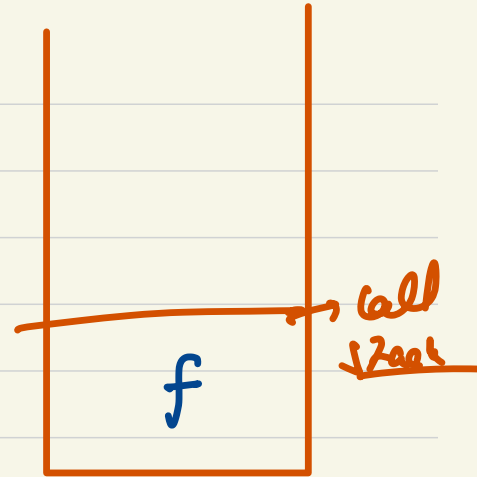
Chrome ✓



Time  
Network  
⋮

```
Part 1 > js AsyncNature0100.js > ...
1 → setTimeout(function f() {
2   console.log('Hello');
3 }, 10);
4
5 → let x = 0;
6
7 → for(let i = 0; i < 10000000000; i++) {
8   x = x + i;
9 }
```

10<sup>9</sup>



Part1 > JS AsyncNartureOfJS.js > ...

```
1 → setTimeout(function f() {  
2   |   console.log('Hello');  
3   | }, 10);  
4  
5 → setTimeout(function g() {  
6   |   console.log('Hello');  
7   | }, 5);  
8  
9 → let x = 0;  
10  
11 → for(let i = 0; i < 10000000000; i++) {  
12   |   x = x + i;  
13   | } |
```

~~empty~~

event  
loop

Call Stack

R.E

timer1 → 10ms → !

timer2 → 5ms → ✓✓

Wow



queue