

Async Javascript

→ Sync means ek ke baad dusra hoga, jb tak ek command complete na ho, dusra shuru nahi hoga

→ Async means, sare kam ek sath shuru honge, jiska answer, pehle aayega wo Jawaab dedega

eg. task 1 → 5 sec
task 2 → 10 sec
task 3 → 15 sec
task 4 → 1 sec

∴ sync = 31 sec

Async = 15 sec.

How to identify ?

→ if you are using
setTimeout,
setInterval,
promises,
fetch,
axios
XMLHttpRequest

Then, it is async js

what is async js?

→ Sometime's your final code is dependant on another's server & we never know when the data will come from that server, ∴ in this case what we cannot do is writing sync code.

∴ to deal with this we write async code & jo bhi code uske respect me hai vo chl jayega.

eg. — sync

setTimeout (function() { }, 2000); async

— sync

you can write the line inside
function, which will run after 2 sec.

o/p:- Sync
Sync

async after 2 sec.

-- js is not asynchronous?

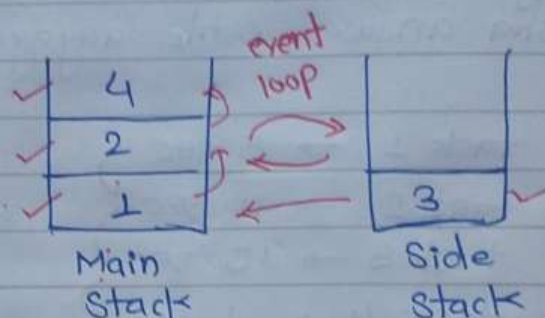
1 — sync

2 — sync

3 — async

4 — sync

There are two stacks



MS → holds sync code

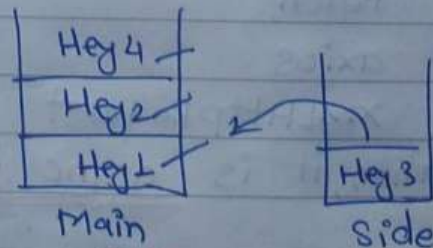
SS → async code

the comm. betn
main & side stack
is maintained by
the event loop.

whenever, the main stack gets empty
then code from side stack is moved
to main stack.

(if code in ss is completed)

eg. console.log("Hey 1")
console.log("Hey 2");
setTimeout (function () {
 console.log("Hey 3");
}, 0);
console.log("Hey 4");



∴ o/p: Hey 1

Hey 2

Hey 4

Hey 3

even if 0 sec. wait is
there, it runs one after another..

--imp

* Single threaded → one computation at a time
multi threaded → multiple " at a time

JS?

-- How to write?

you can request the data by writing/using --

- fetch
- axios
- promises
- setTimeout
- setInterval

sending request

where this data is handled?

after getting
the data

- then catch
- callbacks
- async await

-- callbacks

functions that runs after sometime..

-- promises

↳ just like you make promises in regular life..

promise will either

- in pending state
- resolved
- rejected

store in variable

use .then(callback)
.catch(callback)..

- if the promise is resolved .then is executed
 - if rejected catch is executed..
 - promise is a constructor function
- both then catch accepts a callback,

```
var a = new Promise (function (res, rej) {  
    return res();  
});
```

```
a.then (function () {  
    console.log ("resolved");  
})  
.catch (function () {  
    console.log ("rejected");  
})
```

practice: if the number is below 5 then resolve
if it is above 5 then reject.

classmate

Date

Page

-- promise chaining

do the following tasks one after another..

1. come home then
2. talk with meow then
3. do some coding then..
4. then goto sleep.

Aranshinde

```
var t1 = new Promise (function (res, rej) {  
    return res("come home");  
});
```

```
var t2 = t1.then (function (val) {  
    console.log(val);
```

```
    return new Promise (function (res, rej) {  
        return res("talk with meow");  
    });
```

```
var t3 = t2.then (function (val) {  
    console.log(val);
```

```
    return new Promise ( ...
```

```
    }  
    )
```

-- throttling kisi code ko control karna, no. of executions

area
↓
mouse is scrolling 500 times → throttle → 50 times

-- async await

when you are fetching some info. by sending a request on url..

classmate

Date _____
Page _____

`fetch('https://randomuser.me/') → returns promise..`

```
• then (Function (val) {  
  return val.json();  
})
```

∴ apply then

→ again you need to convert raw data to json ∴ use then

```
• then (Function (val) {  
  console.log(val);  
});
```

→ finally get the data

you can eliminate this process by using **async await**

1. make the nearest parent function **async**
2. **await** the fetch
3. **await** the json data..

↓ **await** removes ↓ .then..

e.g. **async** get function `getUser()` {

```
  var raw = await fetch('url');
```

```
  var data = await raw.json();
```

```
  console.log(data);
```

```
}
```

```
getUser();
```

-- **Concurrency** is me sync code and async code ek sath process ho rha tha, this is concurrency

-- **parallelism** focuses on different processor's & their core's

EcmaScript → js standard..

Some other ES6 concepts..

→ In older js there were 3 types of functions..

① Function statement

```
function sum(a,b) {  
    return a+b;  
}
```

② Function expression

```
const add = function(a,b) {  
    return a+b;  
}
```

③ Anonymous function

```
const square = function(x) {  
    return x*x;  
}
```

④ const double = (num) => num*2;

→ In new js, there is only one function arrow function

① Basic fat arrow function

() => { }

```
var a = () => { }
```

② with one parameter

```
var a = (param) => { }  
a(12);
```

OR

```
var a = param => { }  
a(12);
```

③ with implicit return

```
var g = () => 133;
```

```
console.log(g());  
    ↳ 133 ✓
```

* when you pass multiple arguments then use 1st way..

-- template literals (Backticks)

``...``

classmate

Date

Page

in normal way..

```
console.log("2+2 is", 2+2, " and 2*3 is ", 2*3)
```

↓
but using template literals

```
`2+2 is ${2+2} and 2*3 is ${2*3}`
```

-- default values

```
function abcd (param)
{
  console.log(param);
}
```

abcd(1);

abcd();

o/p: 1
undefined

without default

```
function abcd (param=0)
{
  console.log(param);
}
```

abcd(10);

abcd();

o/p: 10
0

with default..

-- spread & rest

```
var a = [1, 2, 3, 4];
```

```
var b = [...a];
```

↑
spread

copying values..

```
function sum(a, b, ...c)
```

```
{
```

```
}
```

1 2 [3, 4, 5]

↑
Rest

```
a sum(1, 2, 3, 4, 5);
```

when you want to copy
remaining values..

-- destructuring
getting values out from array, objects...

classmate

Date _____
Page _____

```
var arr = [1, 2, 3];  
var [a, b, c] = arr;  
    ↓ ↓ ↓  
    1 2 3
```

```
var obj = {name: "Kiran", age: 22};  
var {name} = obj;
```

↓
"Kiran"

```
var [x, , z] = arr;  
    ↓   ↓  
    1   3
```

-- try catch

```
console.log("Hey");  
console.log(hey);  
console.log("Hey");
```

→ interpreted ... o/p → Hey
→ now, this throws an error
∴ no lines after it
are executed...

```
∴ console.log("Hey");  
[ try {  
    console.log(hey);  
} catch (err) {  
    console.log(err);  
}]
```

o/p: Hey
hey is not defined
Hey..

```
console.log("Hey");
```