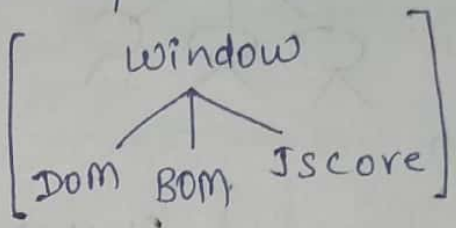
DOM + Modern JS

16/02/23

① Window

Global object can access anywhere, which represents window created by browser.



Topmost hierarchy is window. All methods & property lie in window.

→ It represents a browser window, can control browser window.

Ex ⇒ Window.console.log(—);

② DOM (Document Object Model)

convert HTML code to 'JS object', this is called DOM.

write document in console, for whole HTML code to document to access body;

Ex ⇒ document.body;

[We will learn how we will change HTML codes or CSS code using JS.]

③ BOM (Browser Object Model)

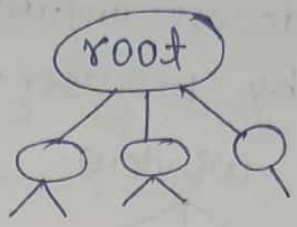
It allows 'JS to talk to browser' about matters other than content of page.

Matters like → location, History, screen,

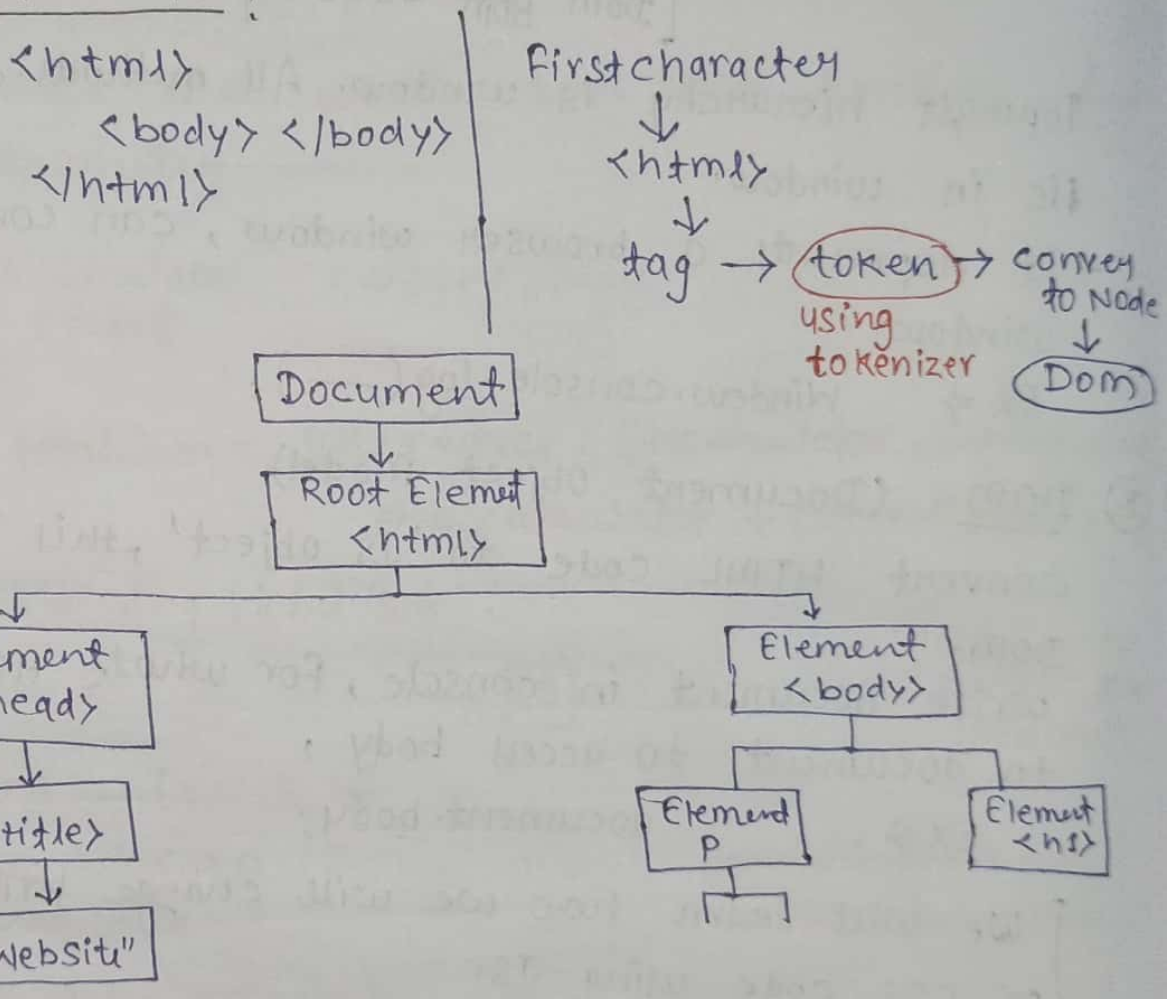
Bom is used to communicate to browser.

* Indepth, DOM

Document Object Model · web page converted to JS. It is like a tree structure



How it render?



* Method to fetch any particular element.

① `getElementById('header');`
↳ id of html tag

↳ It is called on document Object.

↳ It returns a single Object (Because id always unique).

② For multiple

getElementsByClassName(' ')

↳ class of html tag.

↳ Return array-like-object of all child element.

↳[HTML collection interface].

↳ to iterate on Document.getElementByClassName() we use for loop.

③ To Fetch Tag

getElementsByTagName(' ')

↳ return multiple tags of HTML doc.

* NOTE

getElementsByClassName()

&

getElementsByTagName()

① Both method use document object.

② Both return multiple items.

③ The list returned is 'Not an array' its HTML collections.

Trick

Select or hover particular element then in console Write $0 to fetch that particular element.

then we can also put it in variable.

```
let para = $0;
```

We can also fetch className.

para.className

(or)

$0.className

* More Ways

[querySelector() method]

```
let a = querySelector('#header');   → id (only first)
let b = querySelector('.header');   → class (only first)
let c = querySelector('header');    → tag (only first)
```

↳ only return single output first one.

For Multiple Selector

querySelectorAll() method

↳ for all class & tags.

① * Update Existing content of Web page

properties
- ↳ .innerHTML. } get/set HTML content
- ↳ .outer HTML
- ↳ .text content } get/set textual content.
- ↳ .innerText

① .innerHTML

↳ Will try to render HTML tags if written in between.

↳ get an element /all of it descendent HTML content

↳ set an element HTML content.

② .outerHTML

③ ·textContent

↳ tag will also be treated as normal text
↳ this will also show the hidden display

④ ·innerText

↳ This will not show the Display hidden.

②* Adding New Element / content using JavaScript

·createElement()

create
let newChild = document.createElement ('Span');

Add
content.appendChild (newChild);

Ex- let content = document.querySelector ('.class');
let para = document.createElement ('P'); ↳ tag
content.appendchild (para);

paragraph (tag) will be added in above of last tag.

* Creating Text Node

① Ex :-
```
let para = document.createElement ('P');
let text = document.createTextNode ('I am the text');
para.appendChild (text);
content.appendchild (para);
```
<P> I am the text </P>

② Easy way
```
let para = document.createElement ('P');
para.textContent = "I am the text";
content.appendchild (para);
```
↑ last sibling (But)

But, If we want to do positioning of our added element.

insertAdjacentHTML()

↳ has to be called by 2 argument.

① ↳ location / position (where)

② ↳ HTML text / content to be insert (what).

- before begin
  <P>
  - after begin
     ——
  - before begin
  </P>
- after begin

Ex
```
let content = $0;
let newText = document.createElement('h3');
newText.textContent = 'ABCD';
content.insertAdjacentElement('beforeBegin',
                              newText);
```

③ ✱ Remove

.removeChild()

↳ opposite of appendChild()

↳ parent element known.

↳ the child element to remove must be known.

```
parent.removeChild(childElement);
```

Ex
```
let childElement = document.querySelector('.tmpText');
let parentElement = document.querySelector('.parent
                                            Text');
```

parentElement.removeChild(childElement);

Another Way - without parent element deletion

parent = childElement.parent
(To find parent.)

child.parent.remove(child);

— X —

Now, For CSS

Style page content using JS.

- .style
- .cssText
- .setAttribute
- .className
- .classList

} properties we have

Inline CSS → High priority.

① let content = $0;
content.style.color = 'red';
(We can only modify one element with this property)

② content.style.cssText = 'color: green;
background-color: yellow;
fontSize: 4em;'
(Here we can do for multiple properties.)

③ content.setAttribute('style', 'color: Red');
(name, value)

also can add multiple.
also, we can add id, class etc.

content.setAttribute("id", "thisid");

↳ but we are breaking Seperation of concern here to resolve we have other properties.

④ content.className

to get all className of content

↳ will return String.

content.className.trim().split(' ');

will return array of classes.

Its length use (.classlist) will return object (array of classes).

⑤ classList

↳ return Array of classes.

- add()
- remove()
- toggle() → if element not present then, add if present then remove.
- contains → if element present return true if not present will return failse.

18|02|23

* Browser Events

When we will load our code every js code will run, But we want some code to run after some Events.

That's we have **Browser Events**

The announcement by browser.

- ↳ click
- ↳ resize
- ↳ scroll
- ↳ double click
- ↳ load event (Dom & img etc) etc

- events
- respond to event
- Data stored in event
- Stop an event
- Phases / Lifecycle of event.

→ invisible

↳ but to watch by using method

`moniterEvents()`

* Monitor Events - write in console.

```
monitorEvents(document);
```

↓

(Then click on document to see the Events of Website.)

This method will let us see different Events, as they are occuring.

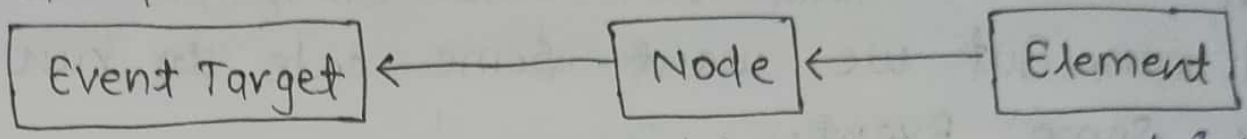
NOTE

- `mointorEvents()` ↳ Turn on the events trigger.
- `unmointerEvents()` → Turn Off

Class are like Blueprint. & Object are Reality

But in js → interface are like blueprint.

↳ addEventListener()
↳ removeEventListener()
↳ dispatchEvent()



↳ Top level interface
No parent
having (3) method
↳ Interface implemented by object that can receive events & may have listener for them

↳ inherit from Event target

↳ inherit from Node (Event target + Node)

* Event listener — Respond for events after Received.
* Node — All method/properties of eventsTarget is inherited by Node.
* Element — Element inherits from Node so, also from Event target.

Event Target —

Interface - Top level entity.

method
↳ addEventListener()
↳ removeEventListener()
↳ dispatchEvent()

① addEventListner()

We can
↳ listen to event
↳ Respond to event
↳ hook into event

Syntax

<event-target>.addEventListner(<event to listen for>, <function-to-run when event happend>)

Two parameter

we Need

① Event - target → on which componet
- → document
- → p
- → div
- → artical etc.

② Event - type
↳ click, double click, scroll etc.

③ function
↳ what to do when event happened.

Ex ⇒
```
document.addEventListener('click', function(){
    console.log('I clicked on Doc');
});
```
↓

Now when you will click the HTML Document
'I clicked on Doc' will be printed in console.

[You can also add it in any particular element rather than whole document & to see change in element.

Ex ⇒
```
let content = document.querySelector('h1');
content.addEventListener('click', function(){
    content.style.background = 'red';
});
```

② removeEventListner()

== Vs ===

↑ loose equality (Value)

Strict equality (value + data type)

Loose Equility → Allows Type corcion when JS will try to convert the items being compared to same type.

Imp → The function you have passed for addEventListner() you need to pass the Exact function to remove EventListner(). we can only Remove when we will create function with name Separately.

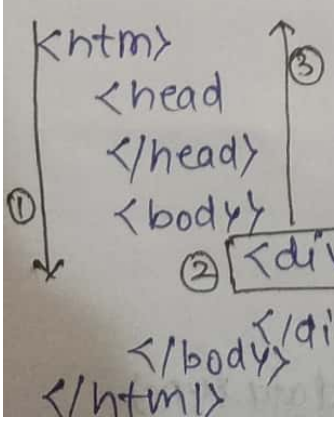
Ex :-
```
function print() {
    console.log('hi');
}
document.addEventListner('click', 'Print);
document.removeEventListner('click', Print);
```

* Why?
When we not create a separtely function then removeEventListner are not work because both function are not same. This is not correct way.

To make removeEventlistner() work
- → Same target
- → Same type
- → Same function.

* Phases of an Event

- → capturing phase → Searching the element where event is triggered
- → At target phase → when reached the element
- → Bubbling Phase ↳returning back from at target phase



Syntax

<event-> .addEventListner (type , listener , usecapture);

<event-> type

type ↑ event type

listener ↑ function ↓ what should happen after event trigger

usecapture ↑ phase in which event is captured (true)

By default → Bubbling Phase

* The Event Object

When an event occurs, addEventListener function get event object.

lots of information about event.

EX:-

```
const content = document.querySelector('#wrapper');
content.addEventListener('click', function(name)
{ console.log(name);
});
```

You can write other name also.

You will get all event information when clicked on element with 'wrapper' id.

* The Default Action

↳ To prevent default Action we use preventdefault() method, we can change the default working of any element.

.preventDefault()

like

anchor tag → link open.

Ex :- let links = document.querySelectorAll ('a');

Now to fetch 3rd from

let thirdlink = links [2];

thridlink.addEventListner ('click', function(event)

{ event.preventDefault ();

console.log ("DONE");

});

→ this will change the default Action of anchor tag.

* How to Avoid Too many Events ?

Ex ⇒ let myDiv = document.createElement ('div');

for (let i=1 ; i<=100 ; i++) {

let newElement = document.createElement('P');

newElement.textContent = 'This is Para' + i;

newElement.addEventListener ('click',

function (event) {

console.log('I have click');

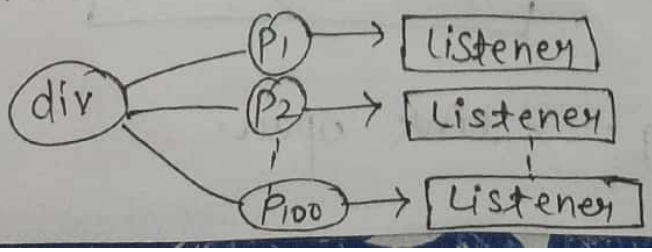
});

myDiv.appendchild (newElement);

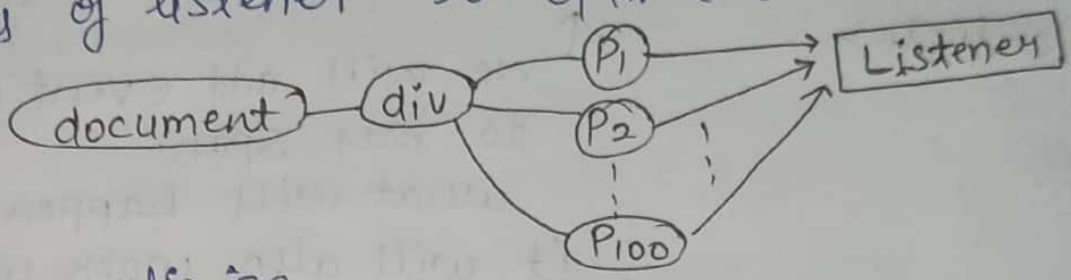
}

document.appendChild (myDiv);

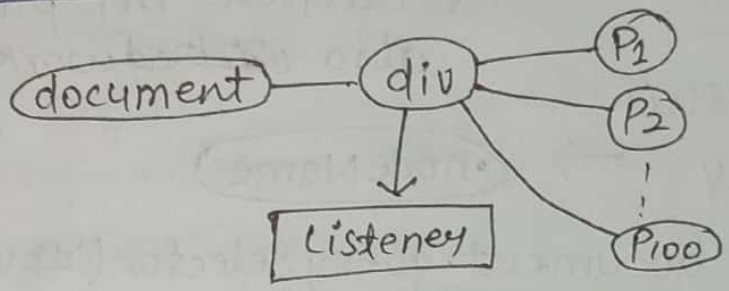
→ created div & created 'P' element in a loop & added event listener to P.



This will take memory as same work & lots of listener to optimize



More optimize



But in this we will not be able to individually access paragraph individually lost. The whole div is accessed.

NOW PHASE WILL HELP HERE!

* Event target Property

The target property return the element where the event occured.

Now the optimised code

Ex ⇒
```
let myDiv = document.createElement('div');
function paraStatus(event){
        console.log('para' + event.target.textContent);
}
myDiv.addEventListener('click', paraStatus);
for(let i=1; i<=100; i++){
        let newElement = document.createElement('p');
        newElement.textContent = 'This is para' + i;
        myDiv.appendChild(newElement);
}
document.body.appendChild(myDiv);
```

Ex `<article id="wrapper">`
`<p> para <span> SPAN </span> </p>`
`</article>`

↑ we will add event listener to this span.
what will happen?
it will also work when (p) is clicked on para, span also ~~clicked~~ work

Now to get of this use property → `.nodeName`

```
let element = document.querySelector('#wrapper');
element.addEventListener('click', function(event){
    if(event.target.nodeName === 'SPAN'){
        console.log('Span clicked' + event.target.
                                    textContent());
    }
});
```

Specific tag Filtering

———X———

* Why `<script>` at the bottom of `<body>` tag
if it will be in head tag script will work before HTML document is loaded.

[How we will know HTML is loaded by
Event → DOMContentLoaded]

if you want to use in head, write dom content loaded event inside script
But Best Practice is bottom of `<body>` tag.

19/02/23

* Performance

↳ measure speed of code.
↳ how to write efficient & performing code
↳ Event loop.

A Standard way to measure how long your code takes to run.

By using method. `Performance.now()`

This is very accurate.

Ex
```
const time1 = performance.now()

// This is you code

const time2 = performance.now()
console.log (time2 - time1);
```

When we add paragraph in DOM 2 things happened
→ Reflow (calculations for element dimension & positioning etc).
→ Repaint (to show element pixel by pixel on your screen)

⇒ Good pratice is → less Reflow & Repaint repetition in your code. Doc.

[Reflow takes more time
Repaint takes time but less than Reflow]

→ Best Pratices → use `Document Fragment`

light weight document object, no reflow & repaint when we add element to it, then we will add document fragment to doc. then it will do one Reflow & Repaint.

* The call Stack

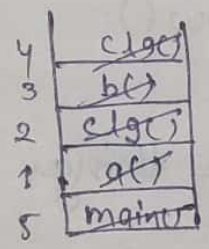
Single-threading :- One command at a time Js in Single-threading language.

↳ Processing of one command at a time.
↳ executes line by line
↳ ignores function but when function is called inside function then again line by line.

NOTE

↳ Run to completion nature of language
↳ Js does not execute multiple line or multiple function at a time.

Call Stack is a list that tracks or store the functi



when it is executed it is removed from callstack

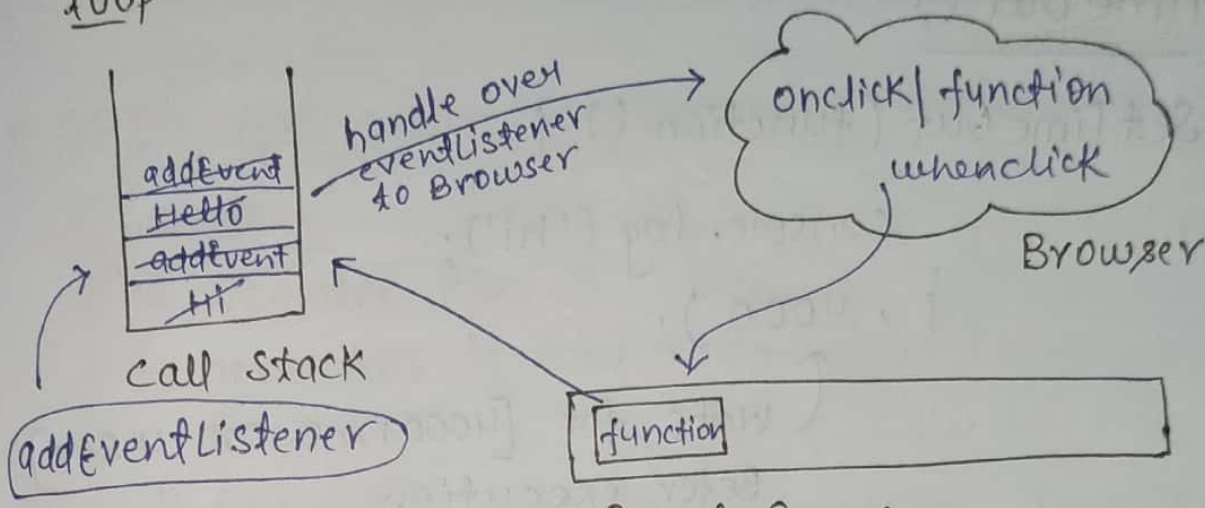
```
function a() {
    clg ('Hi');
    b();
}
function b() {
    clg ('Hello');
}
a();
```

* Event Loop - (Imp)

Synchronous → occuring at a same time.

Asynchronous → EventListner()
↳ becoz it works when action is performed. like - click, scroll etc
↳ SetTimeOut()

Loop



code

```
① console.log('Hi');
  element.addEventListener('click', function(){
                    console.log('123');
  });
② console.log('Hello');
```

Hi
Hello
123

only will run when clicked
③ will be executed

Now, when clicked

Browser will hand over the function to queue, but the Queue will only execute the function, when call stack is empty. If call stack is working on any function, queue will hold the event listener function. when its empty it is executed Finally. This loop is called EVENT LOOP.

A Bit more

① Asynch code – depends on JS Event loop.
② Any Asynch code is handled by browser.

Call Stack → Browser → queue
↕ waits until call stack is empty

* SetTimeOut()

Ex
```
SetTimeOut(function () {
    console.log('hi');
}, 4000);
```

↑ waits for [4000 ms or 4 sec] Befor execution
But no guarantee 4sec is minimum time can take more wait for call stack to be empty.
↳ Because this is also (Async) code.

Explained

SetTimeOut 2 parameter
(function() , time)

When you want to defer (टालना) something, you can use SetTimeOut.

(SetTimeOut, 0)

↳ does not mean to run immediately it will still do the Event loop.

21/02/23

* API :- Application Programming Interface.

Interface

↳ mediator b/w the two

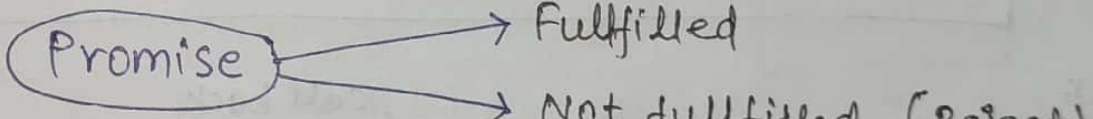
↳ here API is mediatator b/w Frontened & Backend.

↳ Establish the communication b/w two Software components.

* Features of Async Code

- clean & concise
- Better error handling
- easier debugging
-

*



Parallely execute in background in Javascript we use promise.

callback function

Ex =>

```
let meraPromise = new Promise ( function(resolve, reject) {
    console.log ('I am inside promise');
    resolve ('2000');
});
console.log ('hello Pehla');
```

two Parameter

Synchronous

explicitely saying to resolve and pass any value

O/P

- I am inside promise
- hello pehla

Ex ⇒

```
let meraPromise = new Promise (function(resolve,
                                        reject) {
        setTimeOut (function () {
                clg ('I am inside');
        }, 5000 );
        resolve (223346);
});
clg ('hello pehla');
```

Async

O/P

- hlloPehla
- I am inside

NOTE

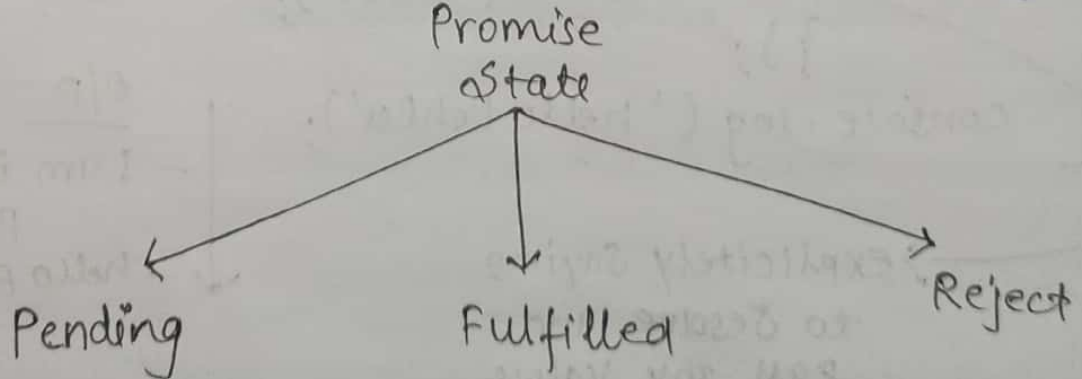
We can also mark reject with an error

```
reject (new Error ('Error Aaya hai');
```

Syntax

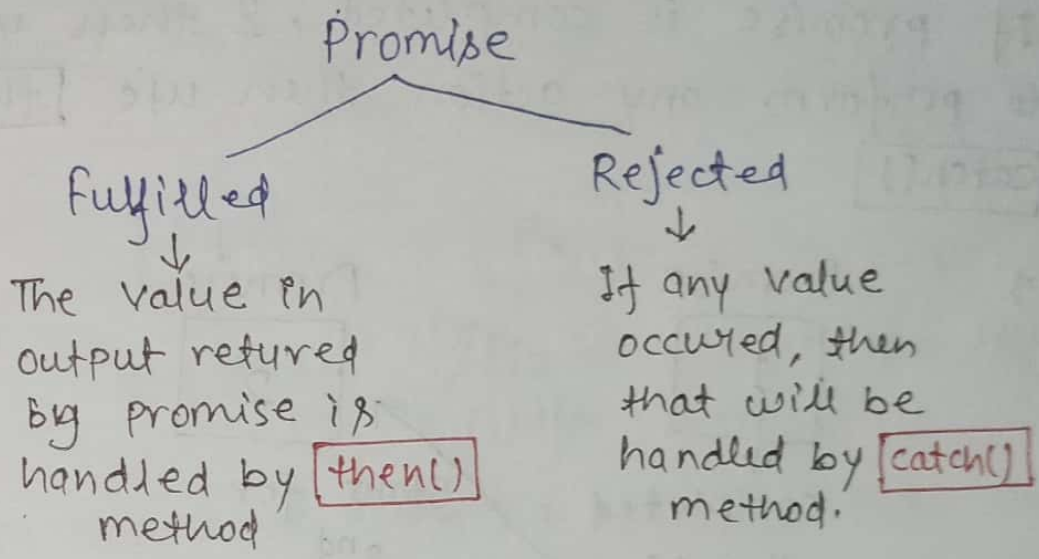
```
let p = new Promise ( call back function) (—— , ——){
        if Succesfully → accepted
        if ,any error → reject
                ↳ So catch the error
                  by explecitly.
}
```

Two Parameter resolve, reject



Promise → represents the eventual completion or failure of an asynchronous operation & its resulting value.

Parallel execution of code using Promise.



After the Promise is done, then we execute anything with the help of

then() → For value or resolve

&

catch() → For error or Reject

Ex

```
let meraPromise = new Promise (function (resolve, reject) {
    SetTimeOut (function() {
        clg ('I am Inside Promise');
    }, 5000);
    //resolve (12345);
    //reject (new Error ('error');
});
meraPromise.then ((value) => {
    clg ('~~error~~ Value') });
meraPromise.catch ((error) => {
    clg ('error') });
```

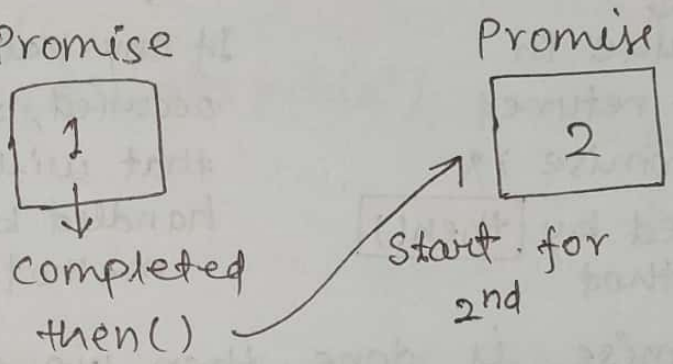
We will give 12345 output

Will give 'error' o/p

⇒ We don't let our Synchronous code wait for asynchronous, We let Asynchronous work in background parallely, we give it promise for accept & Reject of Asynchronous code.

⇒ If promise is completed, & then you want to perform any action then we `then()` or `catch()`

Eg ⇒




```
let Waadaa1 = new Promise (function (resolve,
                                    reject) {
        SetTimeOut (() => {
                clg ('SetTimeOut 1 Start');
        }, 2000);
        resolve (true);
});
let output = Waadaa1 . then(() => {
        let Waadaa2 = new Promise (function (
                                    resolve, reject) {
                setTimeOut (() => {
                        clg ('setTimeOut 2 Start');
                }, 3000);
                resolve ("waada 2 resolved");
        })
return waadaa2;  });
```

Output.then((value) => console.log(value));

X

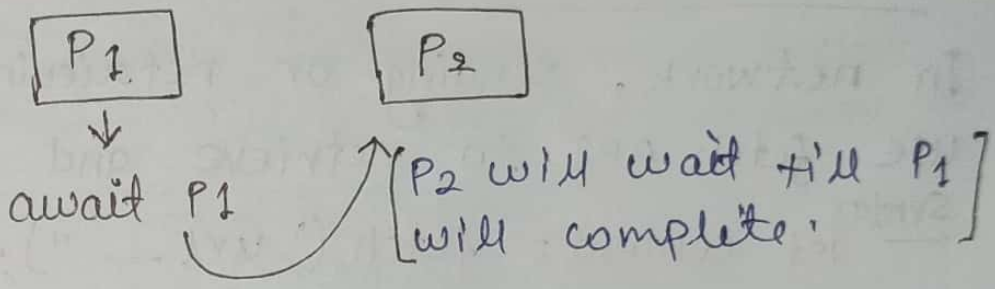
* If we have (so) promise, then so then()?

(No)

↓ use

Async-await → special Syntax used to work with Promise.

use await



when you want to run you code, Async code only when you first, Async code is completed

Ex => To make any code Async

Syntax

```
async function abc(){
        return 7;
}
console.log(abc);
```

→ async will return promise

```
async function utility(){
    let delhiMausam = new Promise((resolve, reject)
        => { setTimeout(() => {
                resolve("Delhi is not");
            }, 5000);
        });
    let hydMausam = new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve("Hyd is cool");
        }, 6000);
    });
```

```
let dm = await delhiMausam;
let hm = await hydMausam;
return [dm, hm];
}
```

→ use await to make it wait else they will run parallely

Fetch API

In network, Sending or retrieving data, we use fetch API to retrieve and to send data.

Syntax
```
let content = fetch (" url --- ");
```
API will return → Promise.

Ex -
```
async function Utility (){
    let content = await fetch (" url ---");
    let output = await content.json ();
    console.log (output);
}
Utility ();
```
↑ Js object Notation

→ data is retrieved here & stored in content & then converted to JSON format.

JSON ⇒ JavaScript Object Notation ie in an object

Key : Value	pair

Fetch API → get () → retrieve

```
let a = fetch (" Url Pair");
```
```
a.status
a.ok
a.JSON ()
a.text ()
```
} to check

Ex
```
let op = a.json();
console.log(op);
```

NOTE

Sometimes the API is protected & you have to send the KEY or your authenticated data (userid), if you want to send then you use 'request header'

```
fetch('url', '[option]')
```
↳ create object & then add authentication or Secret Key.

```
{ header : {
        authentication : key ;
        }
}
```

* Now, Sending using fetch API

Fetch API → post() → Send

↳ fetch along with only url is getcall fetch('url')

↳ fetch along with url & option but the Object in option is secret Key or authentication then also it's get call.

```
fetch('url', 'option');
```

Now, in this option only the way we create object it will be post in which we send data using fetch API.

```
fetch('url', 'option');
```
↳ post

```
let option : {
        method : 'post',
        header :
                :
}
```

POST call Syntax

this object can be copied from inherit

```
async function helper() {
    let option = {
        method : 'POST'
        body   : JSON.stringify({
                    title : 'name',
                    body : 'nikhil',
                    Userid: 2600
                }),
        header : {
                    'content-type' : ---- ';
                }
    }
};
```

We are sending this data in the fetch url to store in database

EX

```
let content = await fetch('url', option){
    let response = content.json();
    return response;
}
async function utility() {
    let any = helper();
    console.log(any);
}
utility();
```

an object is sent in url to update data

Headers is additional information

JSON.Stringify()

↳ converting Object Notation to String format conversion.

———x———

* Now,

Closures

creating function inside function.

Ex =>
```
function abcd(){
    var name = 'xyz';
    function displayName(){
        console.log(name);
    }
    displayName();
}
abcd()
```

O/P

xyz

let is a block scope if we will use let in place of var then also xyz will be printed.

Ex =>
```
let name = "sher";
function init(){
    let name = "NIKHIL";
    function displayName(){
        let name = "Ram";
        console.log(name);
    }
    displayName();
}init();
```

↳ O/P

Ram

variable → var (Global)
→ let (local)

When the function is completed then the name variable will be destroyed.

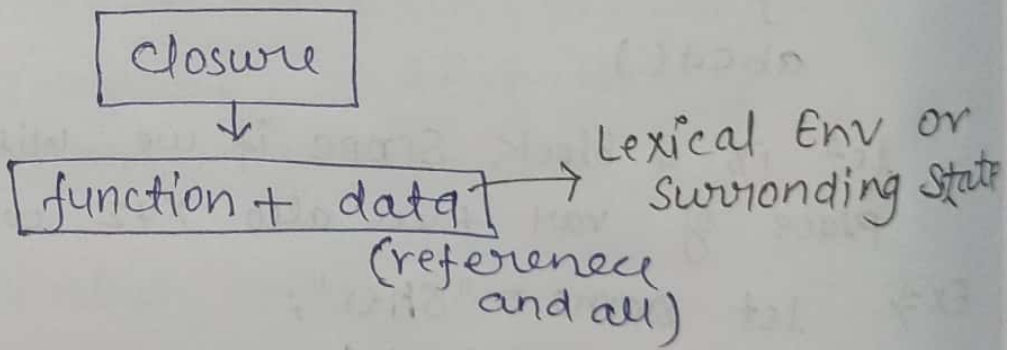
if you will call.

Eg⇒ let funct = init();

funct(); → here name is destroyed but o/p will be 'NIKHIL' (Becoz the closure.)

⇒ When you create nested function every function has its closure.

⇒ Closure is something in which function is binded with its required data.



⇒ With references of data Not copy

⇒ Closure is made for all nested function you create in the form of Reference.

⇒ Nested function → Closure
↓
Reference
NOT COPY