Tuwash Notes

16th February 2023

DOM + Modern JS - class 1

→ DOM
→ BOM
→ window

1) **Window :-** (can access anywhere)

↳ global object which represents window created by browser

Topmost hierarchy is window
all methods & properties lie in window.

→ it represents a browser window, can control browser window

eg. 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, document.body.

// we will learn how we will change HTML codes or CSS codes 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

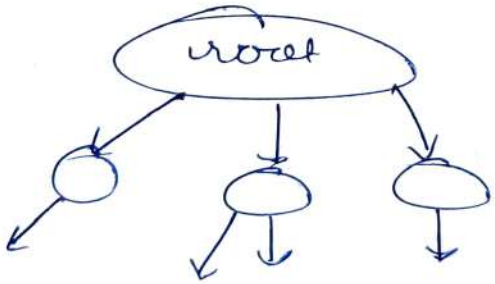
BOM is used to communicate to browser. (like alert)

→ In depth, DOM

Document Object Model.

web page converted to JS.

It is a treelike structure.



How it renders ?

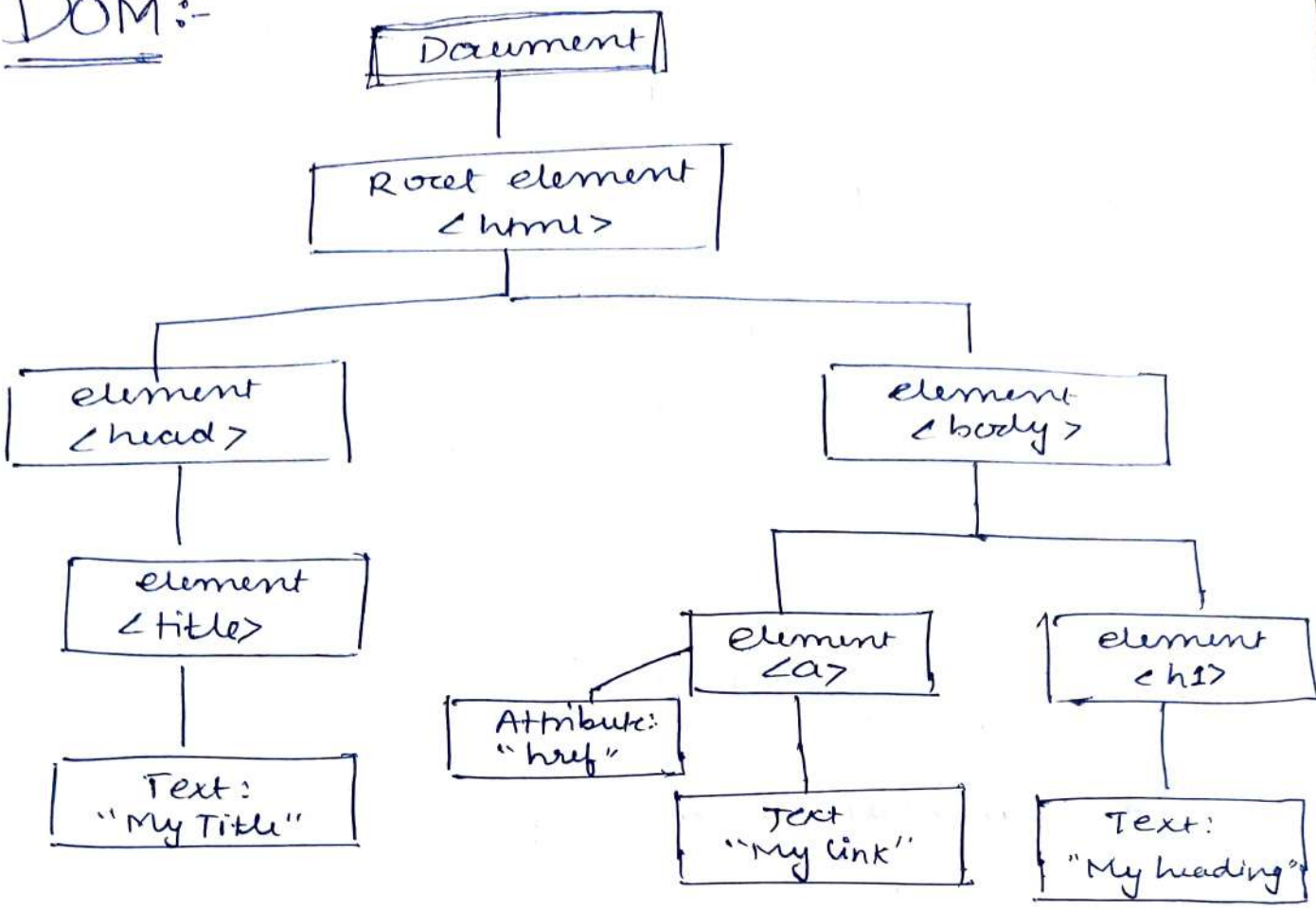
<html>

* First Characters → <html> → then, Tags

Then

Tags → Token (using tokenizer) → converts to Nodes → DOM is ready

DOM :-



Method to Fetch any particular element

↓

getElementById('heading')

↑ id of html tag.

↓

→ it is called on document object

→ It returns a single object (because id is always unique)

For multiple

↳ getElementsByClassName()

→ returns array-like-object of all child elements.

(HTML collection interface)

✓ to iterate on d Document.getElementsbyClassName we use For Loop.

To fetch Tag

getElementByTagName.

↳ return multiple tags of HTML doc.

{ getElementsbyClassName()
&
getElementByTagName() }

↓

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 class Name

{ para.className
or
$0.className }

more ways :-

querySelector() method.

let a = querySelector('#heading'); → Id

let b = querySelector('.heading'); → class (only First)

let c = querySelector('heading'); → tag (only First)

↳ only returns single output First one.

For Multiple Selector

✓ querySelectorAll() method.

↑ for all class & tags.

Update Existing Content of web page

properties.
- .innerHTML — get/set HTML content
- .outerHTML — (H/W)
- .textContent } get/set textual content
- .innerText

1) .innerHTML

→ get an element / all of it descendants HTML content.

→ Set an element's HTML content

innerHTML

↳ will try to render HTML tags if written in b/w.

but

textContent

↳ tags will also be treated as normal text.

↳ this will also show the Hidden Display

innerText

↳ This will not show the 'Display Hidden'

Adding New Element / Content Using JavaScript :-

→ .createElement()

↓

let newchild =
ex:- document.createElement('Span')

↓

(create)

to add

content.appendChild(newchild);

example:-

let content = document.querySelector('.class');

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

P

content.appendChild(para);

paragraph will be tag added

↓

(in absence of last tag).

→ Creating TextNode:-

(1)

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

let text = document.createTextNode('I am the text')

para.appendChild(text);

content.appendChild(para);

 I am the text

② Easy Way

```
let para = document.createElement('p');
para.textContent = "I am the text";
content.appendChild(para);
```

↓

last sibling

But,

If we want to do positioning of our added element

→ insertAdjacentHTML()

- has to be called by 2 argument
- location / position (where) ①
- HTML text / content to be inserted (what) ②

beforebegin
afterbegin
beforeend
afterend

→ add previous Sibling.

— before begin —

— after begin —

 —

— before end —

— after end —

Example :-

```
let content = $0;
let Text = '<h3> Text </h3>
let newText = document.createElement ('h3')
newText.textContent = 'ABCD';
content.insertAdjacentElement ("beforeBegin",
                               newText);
```

Remove

↳ .removeChild()

- ↳ opposite of appendChild()
- ↳ parent element known
- ↳ the child element to remove is must be known.

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

give class to element, then.

```
let childElement = document.querySelector
                   (':tempText');
let parentElement = document.querySelector
                   ('.parentText');
parentElement.removeChild (childElement);
(content)
```

One More Way

↳ without parentElement deletion.

parent = childElement.parent

↳ To find parent.

child.parent.remove(child);

↳ H/W

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;
Font-size: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", "this id");

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

(4) content.className
to get all class Names of content.

↳ will return String.

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

↓

will return array of classes

↑ Its lengthy use .classList
will return object.
(array of classes) ✓

class list

- 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 False

Tuwash Notes

18th Feb 2023

DOM + Modern JS - class 2

→ 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 (etc)
→ Double click
→ Load event (DOM loading etc)

- events — (Invisible) → but to watch by using method (moniterEvents)
- resond to event
- Data Stored in event
- Stop an event
- phases / lifecycle of event.

* MoniterEvents (write in console)

```
moniterEvents(document);
```

(Then click on document to see the Events of website) ✓

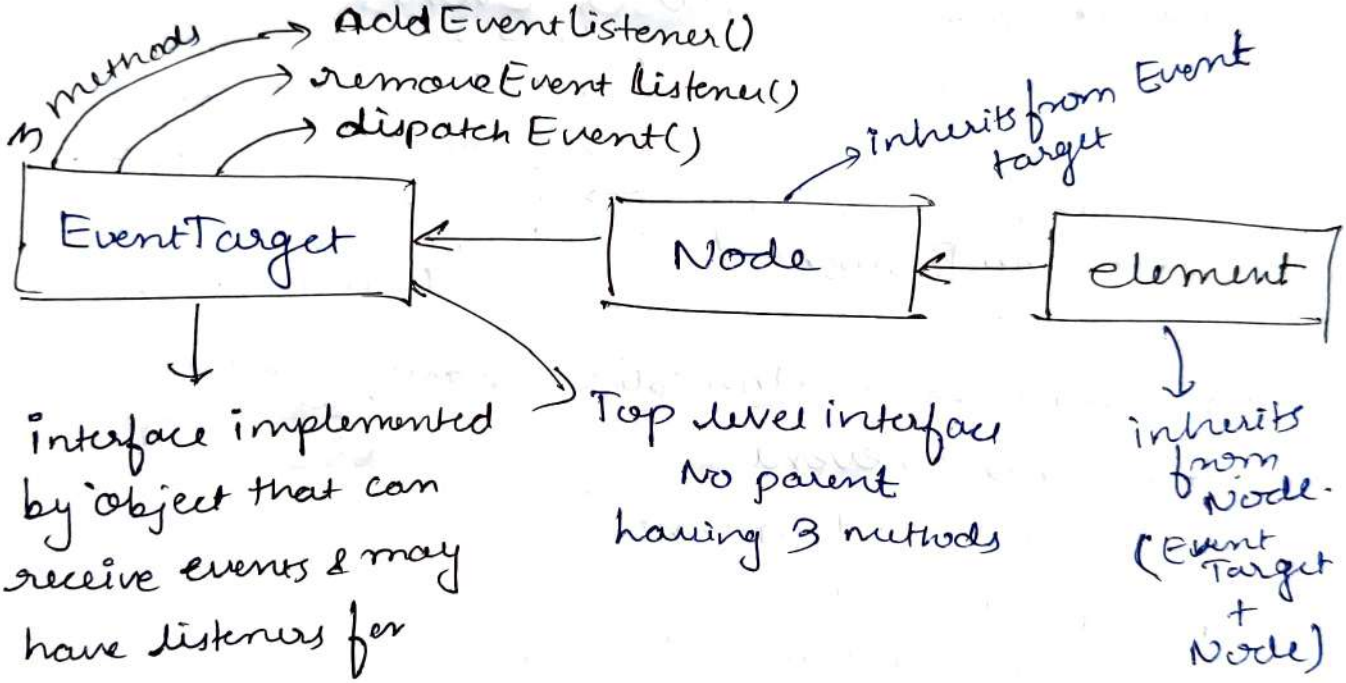
→ This method will let us see diffurent Events, as they are occuring.

monitorEvents()
↳ turn on the events trigger

unmonitorEvents() → Turn off.

{ classes are like Blueprint
& objects are Reality }

in js → interface are like Blueprint



* Event Listener :- Respond for Events after Received

Node :- All methods / properties of EventTarget is inherited by Node

Element :- Element Inherits from Node. so, also from Event target.

Event Target → interface → Top-level-Entity.

Node
elements.

3 methods
- addEventListener()
- removeEventListener()
- dispatchEvent()

1) addEventListener()

we can → listen to event
↳ Respond to event
↳ hook into event

Pseudocode

<eventtarget>.addEventListener(<event to listen for>, <function-to-run when event happened>)

(Two parameters)

We Need
1. Event-target
2. event type → click, double click, Scroll. etc.
3. function → what to do when event happened

on which component
- document
- p
- div
- video
- etc.

eg:- (1) addEventListener

```
document.addEventListener('click', function() {
    console.log("I clicked on Document")
});
```

↓

Now when you will click the HTML Document 'I clicken on Document' will be printed in console

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

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

RemoveEventListener

== v/s ===

↑ loose Equality

↑ Strict equality

allows Type coercion

↳ when JS will try to convert the items ~~big~~ being compared to same type

✓ The Function you have passed for addEventListener you need to pass the Exact Function to removeEventListener.

We can only Remove when we will create function with name Separately.

```
function print() {
    console.log('Hi');
}

document.addEventListener('click', print);
document.removeEventListener('click', print);
```

This will have remove event listener will work because function is an object in javascript. if you will create function in addEventListener & then in remove, they both are not same.

This is not the correct way for removing purpose.

```
('click', function() {

});
```

To make removeEventListener() work successfully

- Same target
- Same type
- Same Function.

You can Chuck any website's event listeners, inspect & then go to ~~Ent~~ event Listeners ~~tab~~ beside Console tab

Phases of an Event :-

- Capturing phase
- At target phase
- Bubbling phase

Capturing phase → Searching of the element where event is triggered

At target phase → when reached the element

Bubbling phase → returning back from at target phase

- Syntax of addEventListener. (3 parameters)

addEventListener(type, listener, useCapture)

- type → event type
- listener → function → what should happen after event triggers
- useCapture → phase in which event is captured. (By default Bubbling phase)

The Event Object :-

When an event occurs, addEventListener function gets event object

↓

lots of information about event

```
const content = document.querySelector('#wrapper');

content.addEventListener('click', function(event) {
    console.log(event);
});
```

- you can write any other name also
- you will get all event information when clicken on element with wrapper id.

The Default Action :-

↳ To prevent default Action we use ~~prevend~~ preventdefault() method.

We can change the default working of any element

- preventDefault()

like:-

anchor tag ⟶ link open

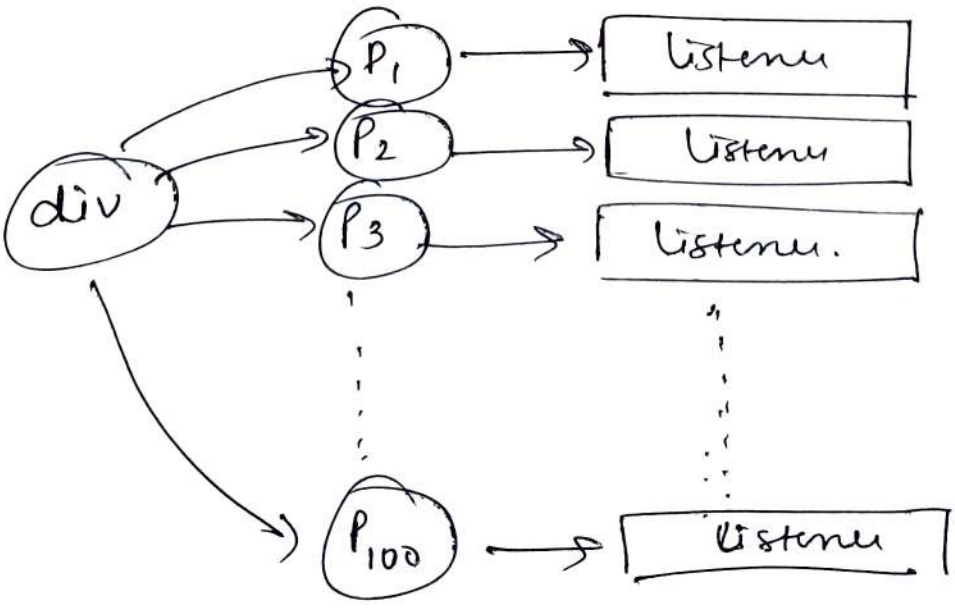
```
let links = document.querySelectorAll('a');
    Now to fetch 3rd from all
let thirdlink = link[2];
thirdlink.addEventlistener('click',
        function(event){
    event.preventDefault();
    console.log('maza aya');
});
```

this will change the default Action of Anchor tag.

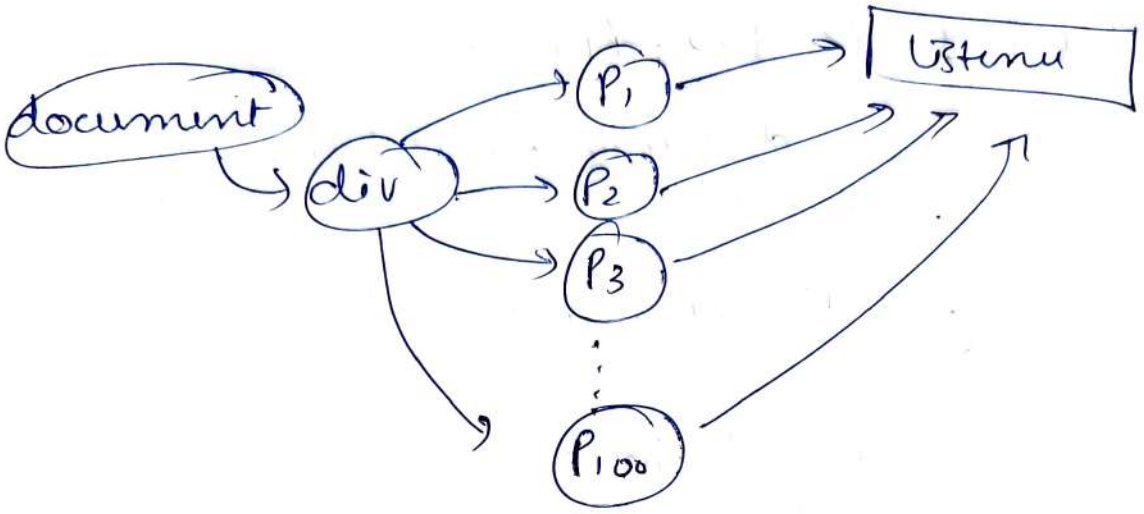
How to Avoid Too many Events ?

```
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 clicked on para');

    });
    myDiv.appendChild(newElement);
}
document.appendChild(myDiv);
```

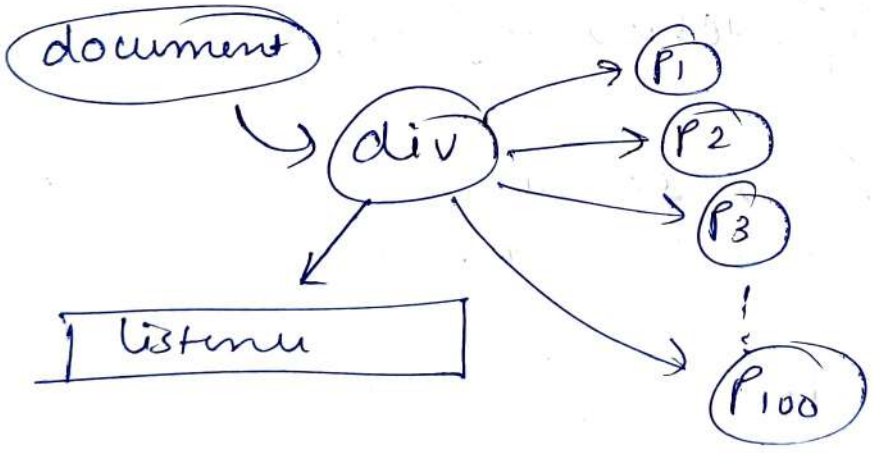
(Created div, & created 'p' element in a loop & added event listener to p.)



This will take memory as
Same work & lots of listeners
to optimize.



More optimize :-



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

NOW PHASES WILL HELP HERE!

Event target property

1) The target property returns the element where the event occurred.

Now the Optimised code :-

```
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);
```

```
<article id="wrapper">
  <p> ABCD <span> xyz </span> </p>
</article>
```

we will add event listener to this span.

what will happen?

it will also work when (p) is clicked ✓ For para, span also works (X)

Now to get rid 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.

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 domContentLoaded event inside Script.

but Best practice is ~~below~~ bottom of <body> tag.

Turwash Notes

19th Feb 2023 ☺

DOM + Modern JS - Class 3

→ 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

performance.now() ← method.

↑ This is very accurate.

```
const time1 = performance.now()

This is your 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 practice is → less Reflow & Repaint repetition in your Doc.

{ Reflow takes more time
& Repaint takes time but less than Reflow }

Best practice

↳ Document fragment

↓

Light-weight document object, no reflow & repaint when we add element to it, then we will add Document fragment to Document Then it will do one Reflow & Repaint ✓

→ The Call Stack :-

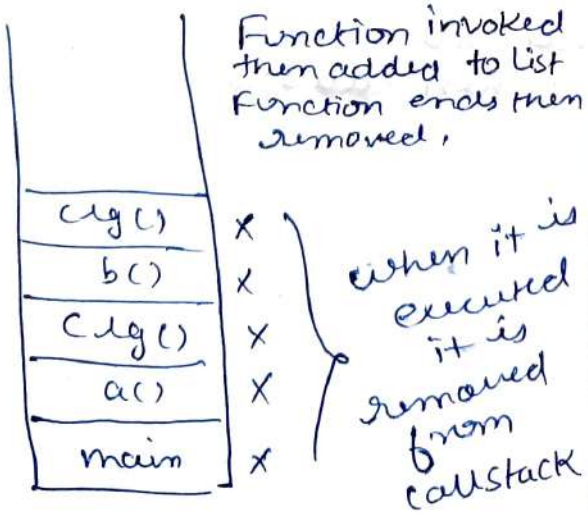
Single-threading :- One command at a time.
JS is Single-threaded language.

✓ processing of one command at a time.

Single-threaded ~~synchronous~~ language :-
executes line by line.
ignores function but when function is called goes inside function then again line by line.

→ run to complition nature of ~~code~~ language.
→ JS does not execute multiple line or multiple function at a time

Call Stack is a list that tracks or stores the Functions :-



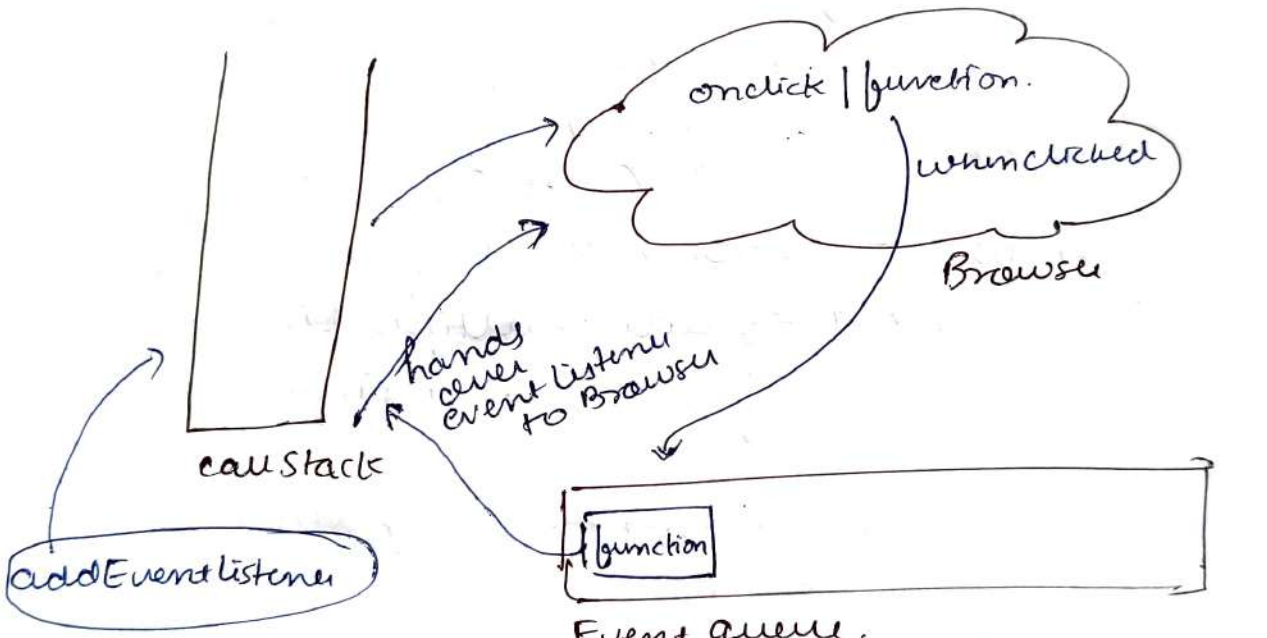

```
function a() {
    console.log('Hi')
        b();
}
function b() {
    console.log('Hello');
}
a();
```

9up Event Loop :-

Synchronous → occuring at a same time.

Event-listener is → Asynchronous.
because it works when action is performed.

Event loop :-



Code :-

```
① clg ('Hi');
② element.addEventListener
    ('click' function() {
    clg ('123');
  });
③ clg ('Hello');
```

② only will run when clicked else ③ will be executed

addEventListener loop EXPLAINED

Code :-

```
1) clg ('ABCD')
2) element.addEventListener ('click', function () {
                clg ('1234');
   }
3) clg ('XYZ').
```

Now, in call stack :-

→ entry of (1) and 'ABCD' is printed & (1) is executed

→ Then entry of (2) event listener. but it is when clicked then function, so, callstack hands over event listener to browser & move to (3). clg ('XYZ')

Now, when clicked

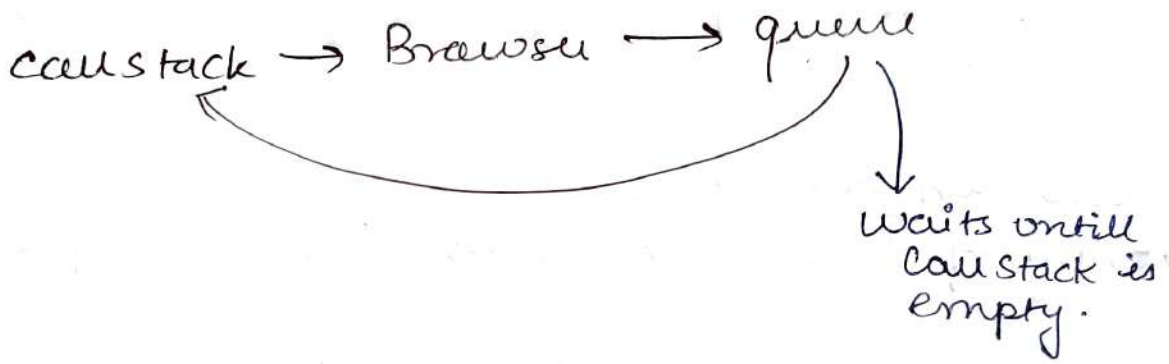
Browser will hand over the function to queue, but the queue will only execute the function, when callstack is empty. if call stack is working on any function, queue will hold the event listener function. when its empty, it is executed Finally.

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.



SetTimeOut()

```
setTimeout( function () {
    clg ('Hi');
}, 4000);
```

↑ waits For 4000ms or 4 sec before execution.

But no guarantee 4sec is minimum time can take more., waits for call Stack to be empty.

↑ because this is also Async Code

SetTimeout & parameter

(function(), Time)

When you want to defer Something, you can ~~you~~ use SetTimeOut.

SetTimeout, 0

↓

does not mean to run immediately.

it will Still do the Event Loop.

Tuwash Notes

21st Feb 2023

DOM + Modern js - Class 4

→ API :- (Application programming Interface)

Interface

↳ mediator b/w the two

here API is mediator b/w Frontend & Backend.

Establish the communication b/w two Software Components

Features of Async Code

- Clean & concise
- Better error handling
- easier debugging
- H/w

Promise :-

- Fulfilled
- Not Fulfilled

✓ parallely execute in background in javascript we use promise

Async promise

call back function

two parameters

```
let meraPromise = new Promise (
    function (resolve, reject) {
        console.log ("I am Inside promise");
        resolve (1998);
    });

console.log ('Pehla');
```

output

- I am Inside promise
- Pehla

Now Async

```
let mera Promise = new Promise ( function
        (resolve, reject) {
    SetTimeout ( function () {
        alg ('I am Inside');
    }, 5000);
    resolve (2233);
});
alg ('Pehla');
```

→ explicitly saying to resolve

output

- pehla
- I am Inside

We can also mark reject with an error.

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

Call back function

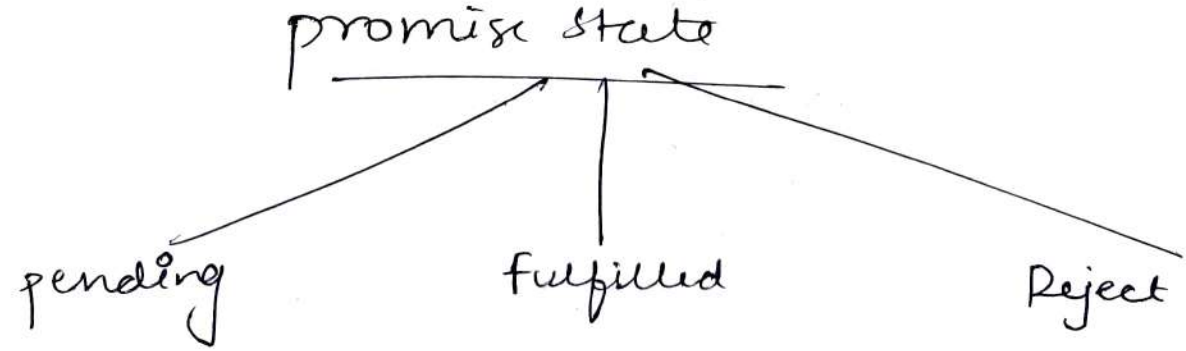
```
let p = new promise ( _____ )
                    (_ , _____ )
{
```

Two parameter (resolve, reject)

if successfully → accepted
if not, error → Rejected.

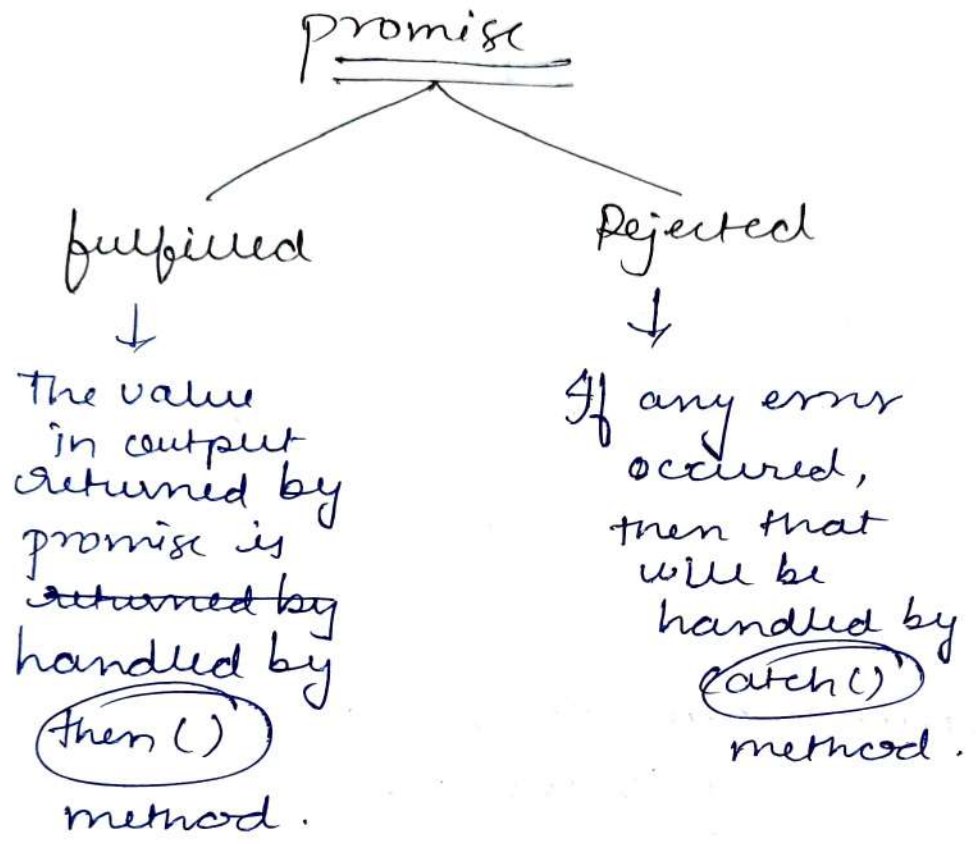
So catch the Error.

```
}
```

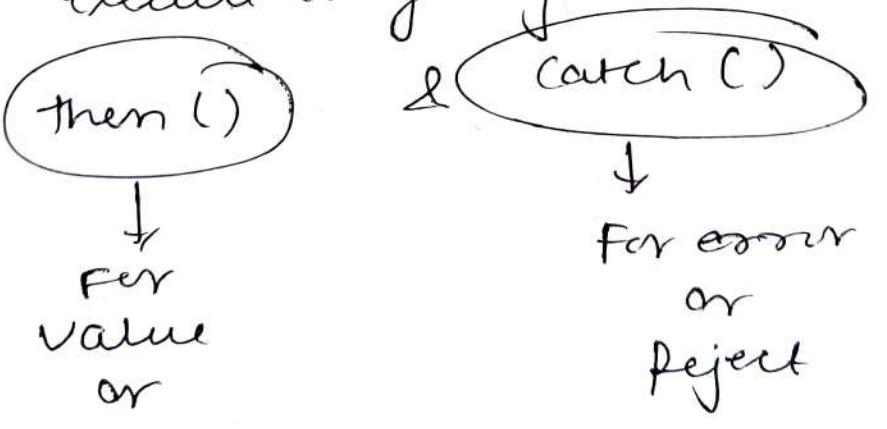


promise:- represents the eventual completion or failure of an asynchronous operation & its resulting value.

✓ parallel execution of code using promise



After the promise is Done, then we execute anything with the help of



```
let newPromise = new Promise (function
                    (resolve, reject) {

setTimeOut (function () {
    clg ('I am inside promise');
}, 5000);

//resolve (12345);
// reject (new Error ('error'))

});

newPromise.then ((value) => {
        clg (value)});
```

↓ will give 12345 output.

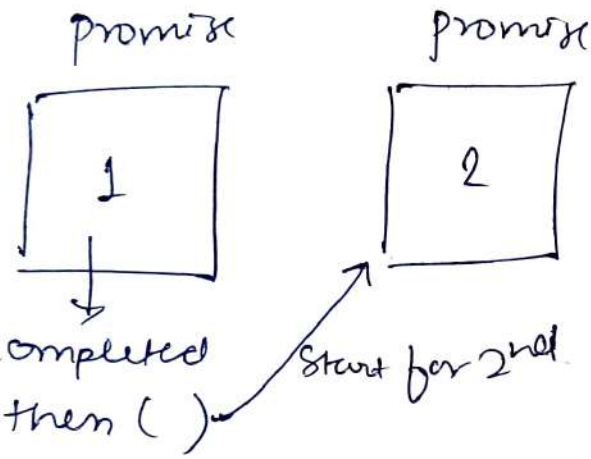
```
newPromise.catch ((error) => {
        clg ('error')});
```

↓ will give 'error' output written. after reject error occurred

✓ We Don't let our Synchronous code wait for Asynchronous, so let Asynchronous work in background parallly, we give it promise for accept & Reject of Asynchronous code

✓ if promise is completed, & then you want to perform any action, then use then() or catch()

eg:-



eg:-

```
let waadaa1 = new Promise (function (resolve,
                                    reject) {
    setTimeout(() => {
        console.log ('SetTimeout1 Started');
    }, 2000); resolve (true);
})

waadaa1. then (() => {
    let waadaa2 = new Promise (function (
                                    resolve, reject) {
        resolve ("waada 2 resolved");
    })
```

here one more promise with 3000

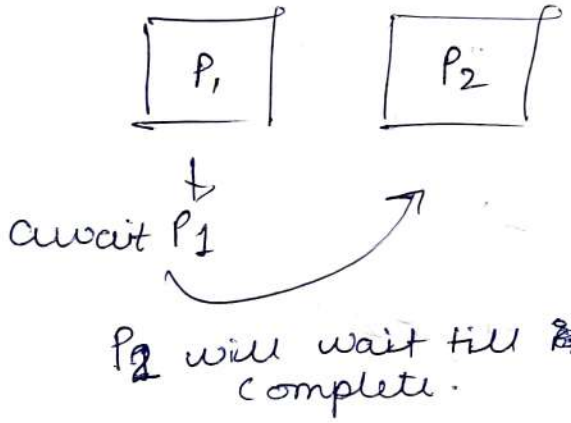
return waadaa!;

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

If we have 50 promises, then 50 then()?

NO

Async-await → Special syntax used to work with promises



When you want to run your another Async code only when your first Async code is completed, use await

---

To make any code Async

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

async will return promise

```
async function utility() {
  let delhiMausam = new Promise((resolve, reject) => {
      setTimeout(() => {
          resolve("Delhi is hot");
      }, 5000);
  });
  let hydMausam = new Promise((resolve, reject) => {
      setTimeout(() => {
          resolve("Hyderabad is cool");
      }, 6000);
  });
  let dM = await delhiMausam
  let hM = await hydMausam.
  return [dM, hM];
}
```

use await to make it wait else they will run parallelly.

Fetch API

In Network,
sending or retrieving data.
we use fetch API to retrieve
and ~~API~~ to send data.

```
let content = fetch("url....");
```

Syntax to fetch API

API will return → promise.

```
async function utility(){
let content = await fetch("url....");
let output = await content.json();
    console.log(output);
}
utility();
```

JavaScript Object Notation.

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

JSON :- JavaScript Object Notation
i.e. in an object
key : value pair

(get call in API)

Fet API → get() → retrieve

↓

let a = fetch (" url pass ");

a.status
a.ok
a.JSON()
a.text()
} to check.

ex:-
```
let op = a.Json();
console.log(op);
```

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

fetch (`url` , '[options]')

↓
create object & then add authentication or secret key.

{ header : {
authentication : key;
}
}

Now Sending using fetch API

post → Send

→ fetch along with only url is get call

fetch ('url')

→ fetch along with url & options but the object in option is secret key or authentication then also its get call.

fetch ('url', 'options')

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

fetch ('url', 'options')

↳ post

let options = {

method : 'post'

header :

:

:

:

}

post call :-

```
async function helper() {
    let options = {
        method : 'POST'
        body : JSON.stringify ({
            title : 'foo',
            body : 'bar',
        }),
        headers : {
            'content-type' : ------';
        },
    };
let content = await fetch ('url', options);
    let response = content.json();
    return response;
}

async function utility() {
    let ans = helper();
    console.log(ans);
}
utility();
```

this object can be copied from internet

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

an object is sent in url to update Data.

headers is additional information

JSON.Stringify()

↳ converting object Notation to String.

Format conversion ✓

Now,

Closures :-

creating function inside function

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

↓ xyz printed

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

```
let name = "Sher";
function init() {
    let name = "Mozilla";
    function displayName() {
        let name = "Babbar"
        console.log(name);
    }
    displayName();
}
init();
```

→ Babbar will be printed

Variable → local → let → block scope

Variable → Global → var

when the function is completed then the name variable will be destroyed

if you will call

```
let funct = init();
funct();
```

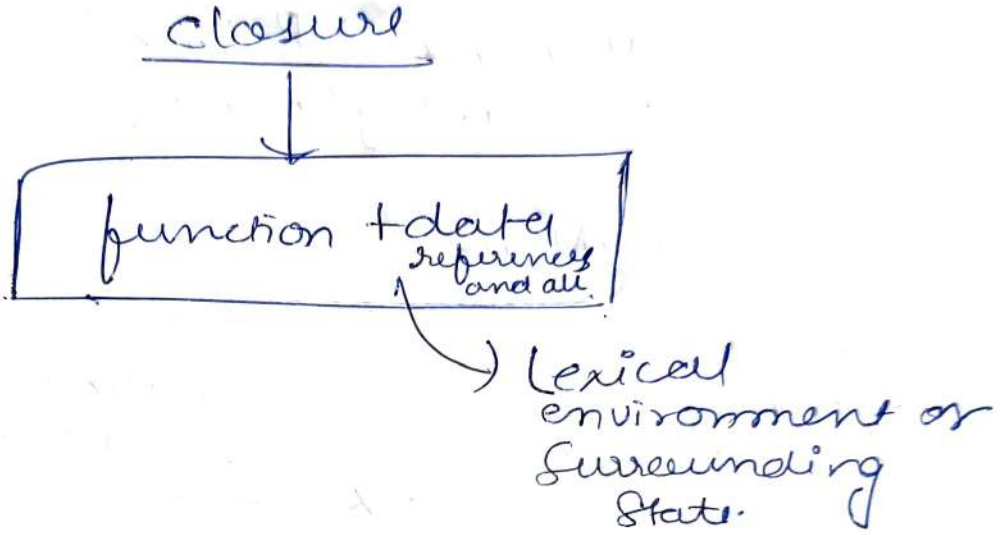
→ here name is destroyed. but output will be 'Mozilla'

↑

(because of 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 References ✓

Nested function → Closure

↓

Reference

Not Copy