

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
 e.g. `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. //

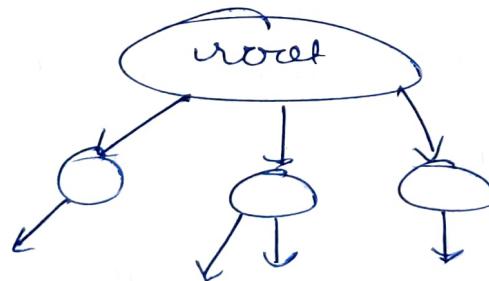
3) BOM :- Browser Object Model

→ 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 tree-like structure.



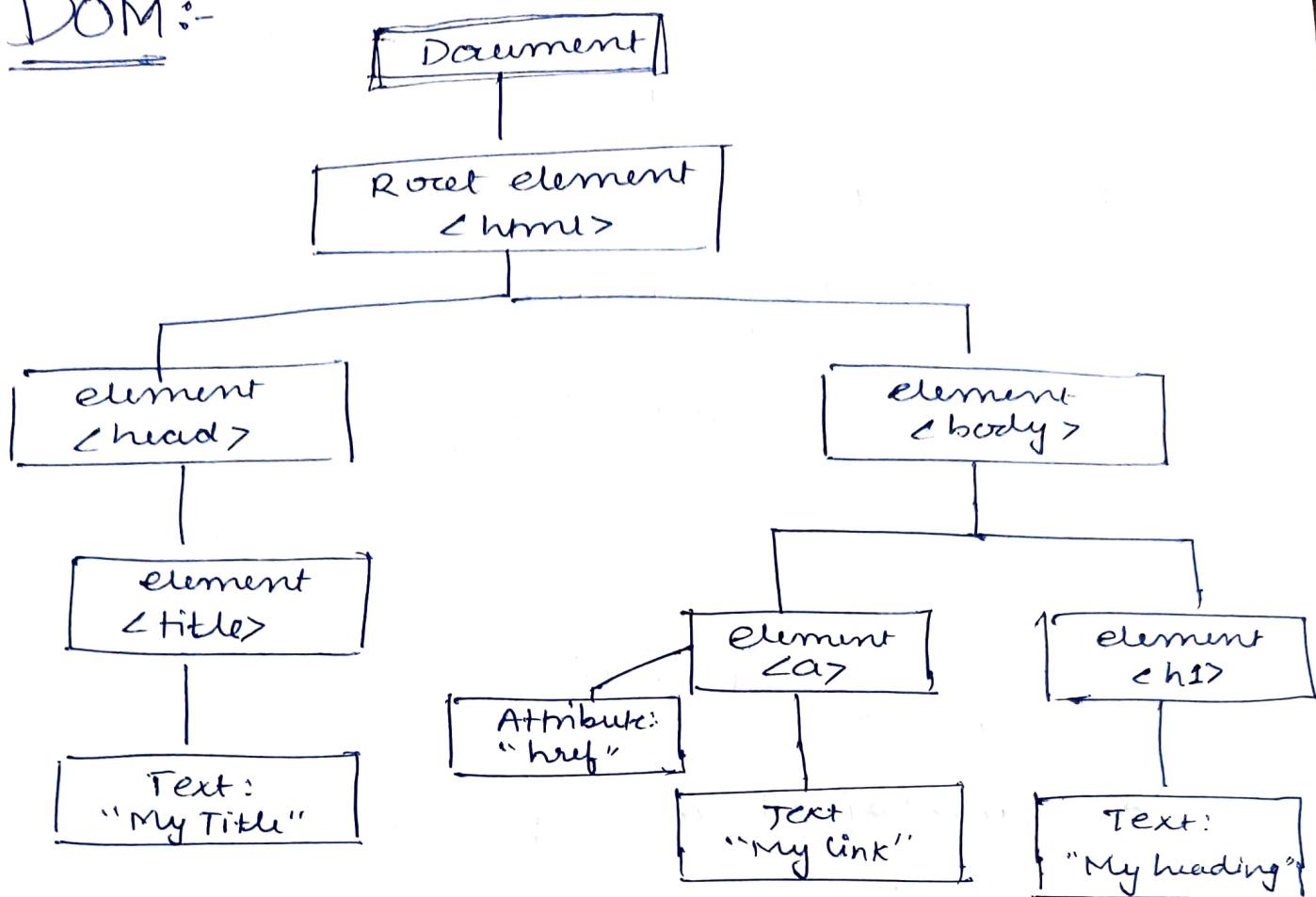
How it renders?

<html>

* First character → <html> → then, Tags
Then

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

DOM :-



Method to fetch any particular element



getElementbyId('heading')

→ it is called on document Object

↑ id of html tag.

→ 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 `Document.getElementById` we use `for loop`.

To fetch Tag

`getElementsByClassName`.

↳ return multiple tags.
of HTML doc.

`getElementsByClassName()`
&
`getElementsByClassName()`



- 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('#header'); → Id

let b = querySelector('.header'); → class (only first)

let c = querySelector('header'); → 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 its descendants HTML content.
- Set an element's HTML content

innerHTML

- ↳ will try to render HTML tags if written in blw.

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');

content.appendChild(para);

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

① → Creating TextNode :-

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

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

my 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,

If we want to do positioning of our added element

→ insertAdjacentHTML()

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

{ beforebegin } → add previous sibling.
afterbegin
beforeend
afterend

— before begin —

<p>

— after begin —

<div> — </div>

— before end —

</p>

— 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
[]
V
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", "thisid");`

↳ 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

↑ gets length 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

→ Implementation

→ Implementation of class list

→ Implementation of add

→ Implementation of remove

→ Implementation of toggle

→ Implementation of contains

→ Implementation of class list

18th Feb 2023

Turwah Notes

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
→ Double Click
→ Load event
(DOM & img etc)

- events → Invisible → but to watch by using method monitorEvents,
- respond to event
- Data stored in event
- Stop an event
- phases / lifecycle of event.

* MonitorEvents

(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 occurring.

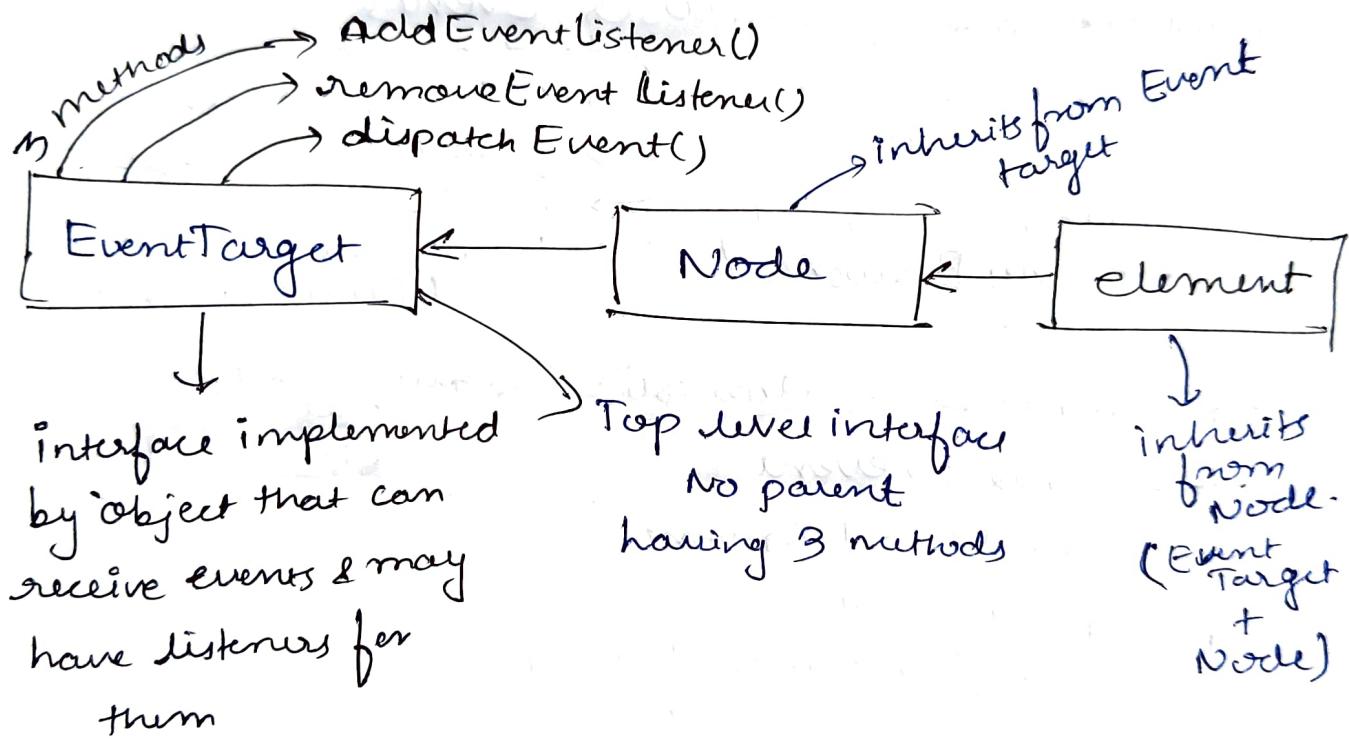
monitorEvents()

↳ turn on the events trigger

unmonitorEvents() → Turn off.

{ classes are like Blueprint
& objects are Reality }

in JS → interface are like Blueprint

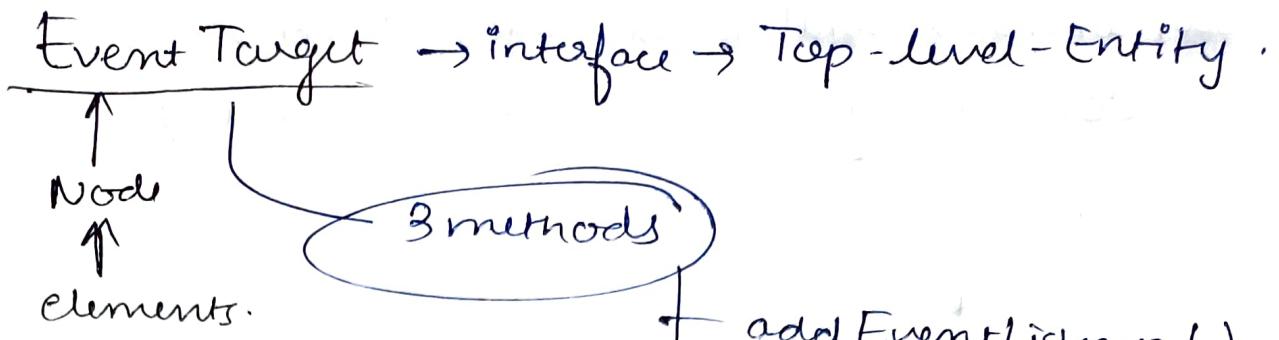


* Event Listener:- Respond for Events after Received

Node:-

All methods &/ properties of EventTarget is inherited by Node

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



1) addEventListener()

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

Pseudocode

(Two parameters)

<event-target>.addEventListener(<event to listen for>,

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
+ div
+ video
etc.

Eg:- ① Add Event Listener

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



New when you will click the
HTML Document.

'I clicked 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.backgroundColor = 'red';  
});
```

Remove Event Listener

= = v/s = = =
q q

loose
Equality

Strict equality

Allows Type coercion

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

✓ The Function you have passed for addEventListner you need to pass the Exact Function to removeEventListner.
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 how
remove event listener
will work
because function is an
object in javascript. if
you will create function
in addEventListner & 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 check any website's event listeners, inspect & then go to ~~event~~ event listeners tab beside Console tab

Phases of an Event :-

- Capturing phase
 - At target phase
 - Bubbling phase
- Searching of the element where event is triggered
- When reached the element
- ↓ returning back from at target phase

- Syntax of addEventListener. (3 parameters)

addEventlistener(type, listener, useCapture)

event type function phase in which event is captured.
what happens after event triggers
(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('div#wrapper');
```

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

```
    console.log(event);
```

you can write any other name also

```
}
```

↑
you will get all event information

when click on element with wrapper id.

The Default Action :-

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

We can change the default
working of any element

• preventDefault()

like:-

anchor tag → link open

let links = document.querySelectorAll('a');

Need to fetch 3rd from all

let thirdLink = link[2];

thirdLink.addEventListener('click',
function(event){

event.preventDefault();

console.log('maza aya');

});

}) ;

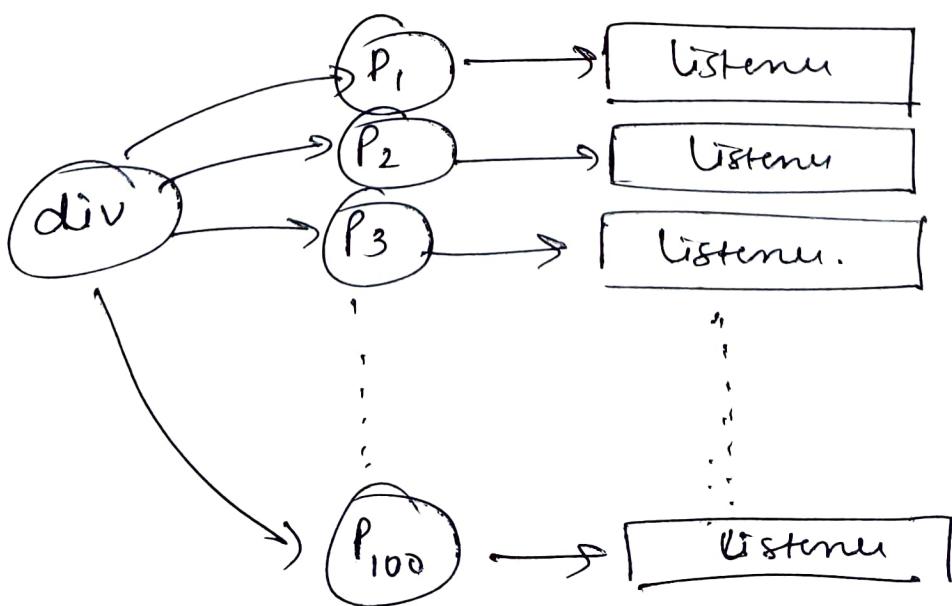
This will change
the

default Action
of Anchor tag.

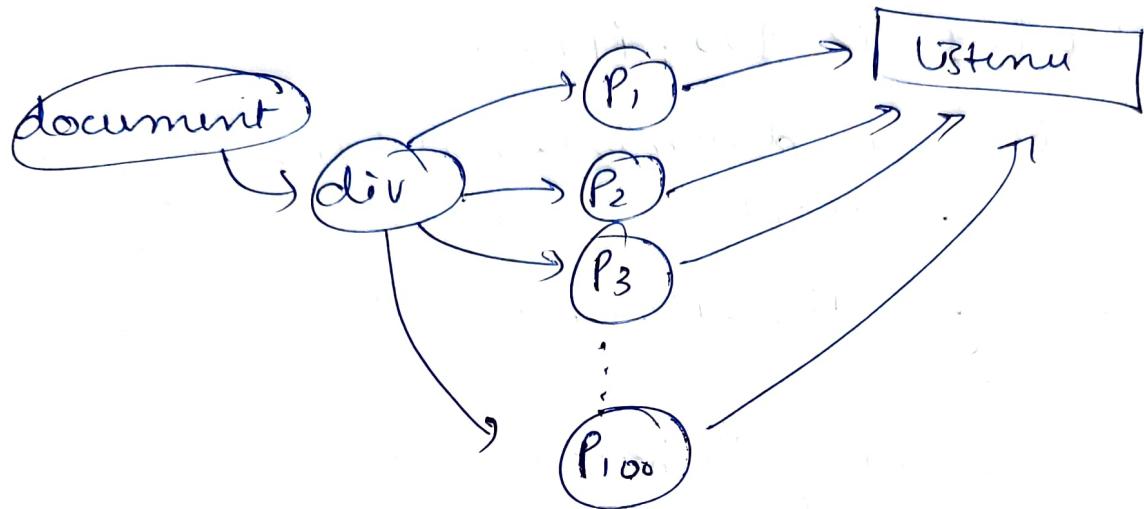
How to Avoid Too many Events {

```
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);
```

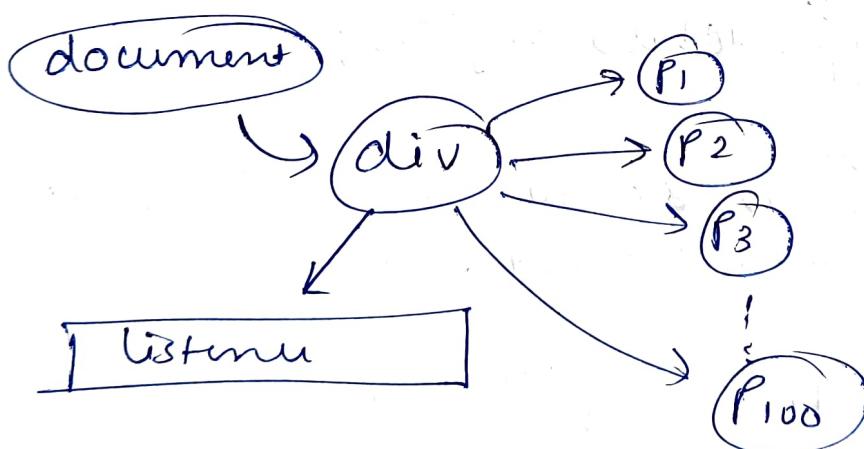
(created div, & created 'p' element in a)
(script & added event listener to p.)



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
individually bcz. The whole
div is access.

Now PHASES WILL HELP
HERE!

Event target property

↳ The target property returns the element where the event occurred.

New 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);
```

----- X ----- X -----
<article id="wrapper">

<p> ABCD xyz </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



Now to get rid of this
use property → nodeName

let element = document.querySelector('#wrap')
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.



19th Feb 2023

Turwah Notes

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()
```

The 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~~ less Reflow & Repaint repetition in your Doc.

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

Best practice

Document fragment

↓
Lightweight 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.

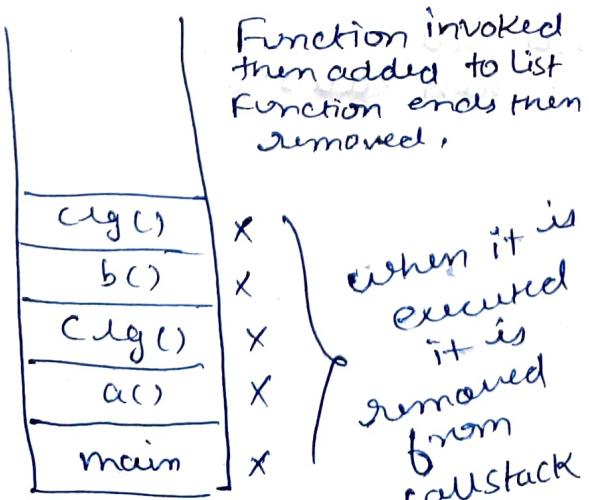
✓ processing of one command at a time

single-threaded
~~Simultaneous~~ language
executes line by line.
ignores function but when function
is called goes inside function then
again line by line.

→ run to completion 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();
```

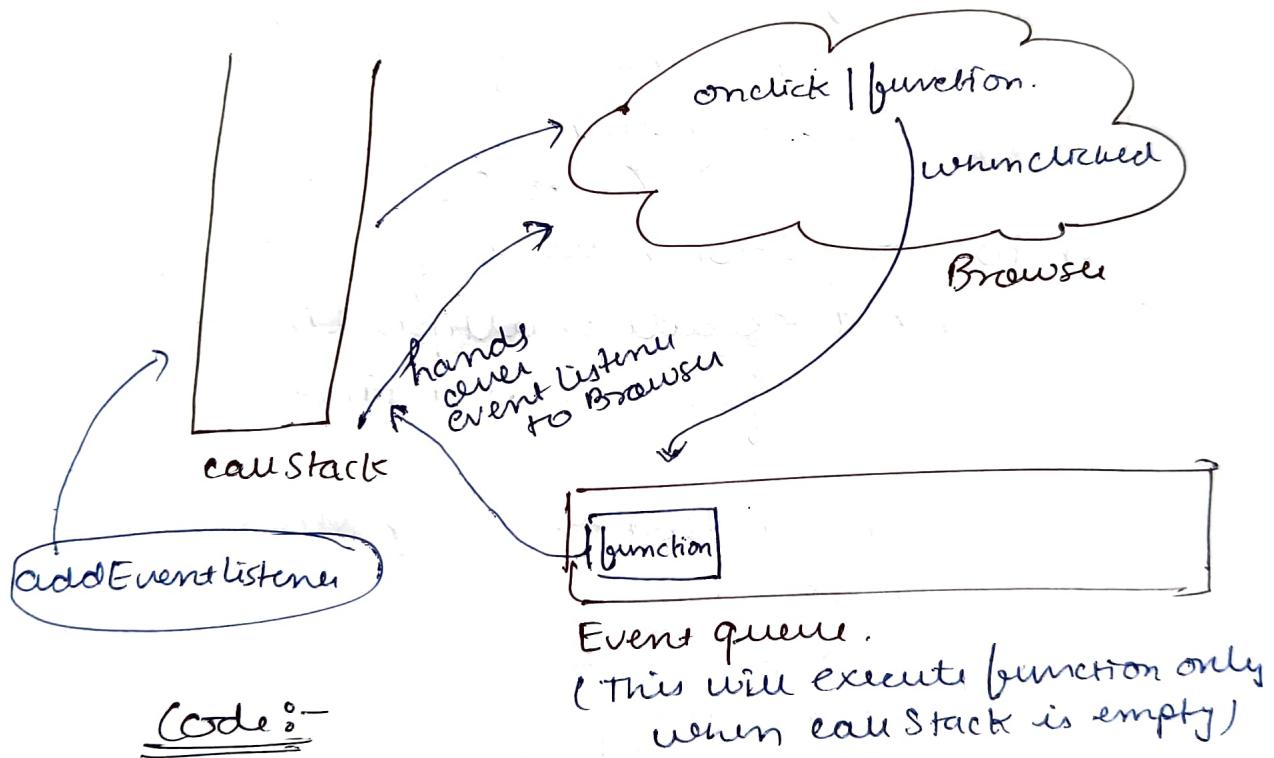
GMP

Event Loop :-

Synchronous → occurring at a Same Time

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

Event loop :-



Code :-

- ① `ug ('Hi');`
 - ② `element.addEventListener('click', function() {
 ug ('123');
});`
 - ③ `ug ('Hello');`
- Only will run when clicked
else ③ will be executed

addEventlistener loop

EXPLAINED

Code :-

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

Now, in call stack :-

→ entry of ① and 'ABCD' is printed & ~~step ①~~ is executed

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

Now, when clicked

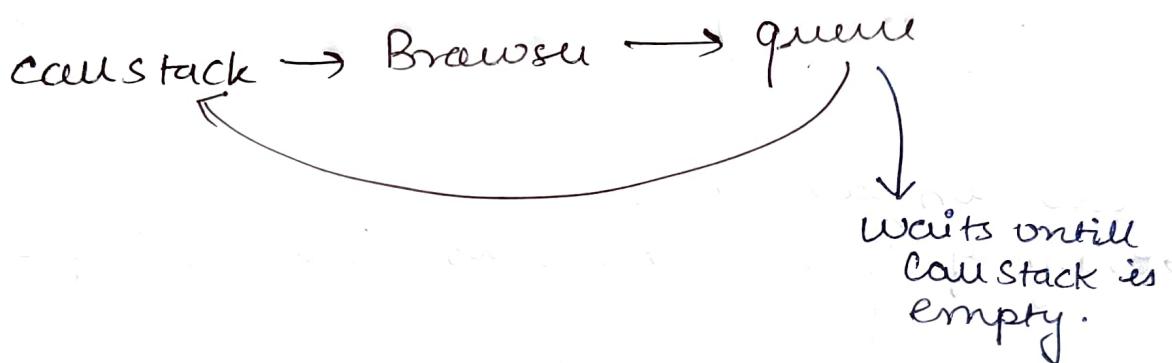
Browser will hand over the function to queue, but the queue will only execute the function, when callstack is empty. if callstack is working on any function, queue will hold the event listener function. when its empty, it is executed finally.

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 () {

 deg ('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.



DOM + Modern JS - Class 4

→ API :- (Application programming Interface)

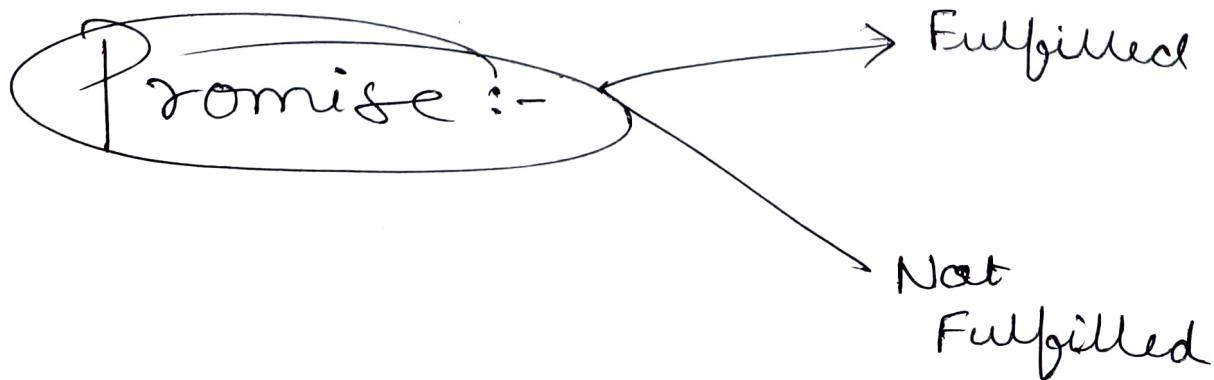
Interface

↳ 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 be debugging
- H/w



✓ parallelly execute in background
in javascript we use
promise

Async promise

```
let myPromise = new Promise (  
    function (resolve, reject) {  
        console.log ('I am Inside promise');  
        resolve ('998');  
    },  
    console.log ('Pehla');  
);
```

call back function
two parameters

Output
- I am Inside promise
- Pehla

New Async

```
let myPromise = new Promise (function  
    (resolve, reject) {  
        setTimeout (function () {  
            console.log ('I am Inside');  
            reject ('5000');  
            resolve ('2233');  
        }, 5000);  
        console.log ('Pehla');  
    },  
);
```

→ explicitly saying
to resolve

Output
- pehla
- I am Inside

✓ We can also mark reject with an error. ↓

```
reject (new Error (' Error Aya'))
```

Call back function

```

graph TD
    A["let p = new Promise (") --- B["( )"]
    A --- C["{"]
    C --- D["( - , - )"]
    D --- E["Two parameters"]
    E --- F["if successfully → accepted (resolve, reject)"]
    E --- G["if not, error → Rejected."]
    G --- H["so catch the Error."]
  
```

let $p = \text{new Promise} (\underline{\hspace{2cm}})$

\downarrow

$(\underline{\hspace{1cm}}, \underline{\hspace{1cm}})$

\uparrow

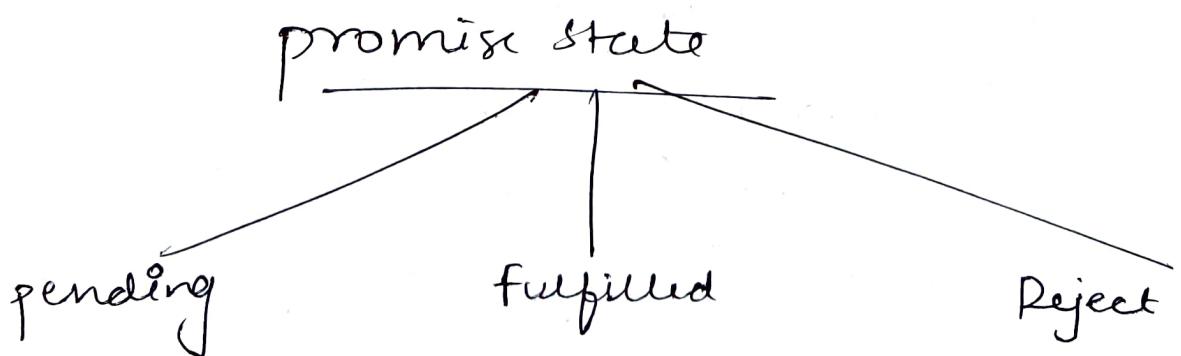
Two parameters

if successfully \rightarrow accepted (resolve, reject)

if not, error \rightarrow Rejected.

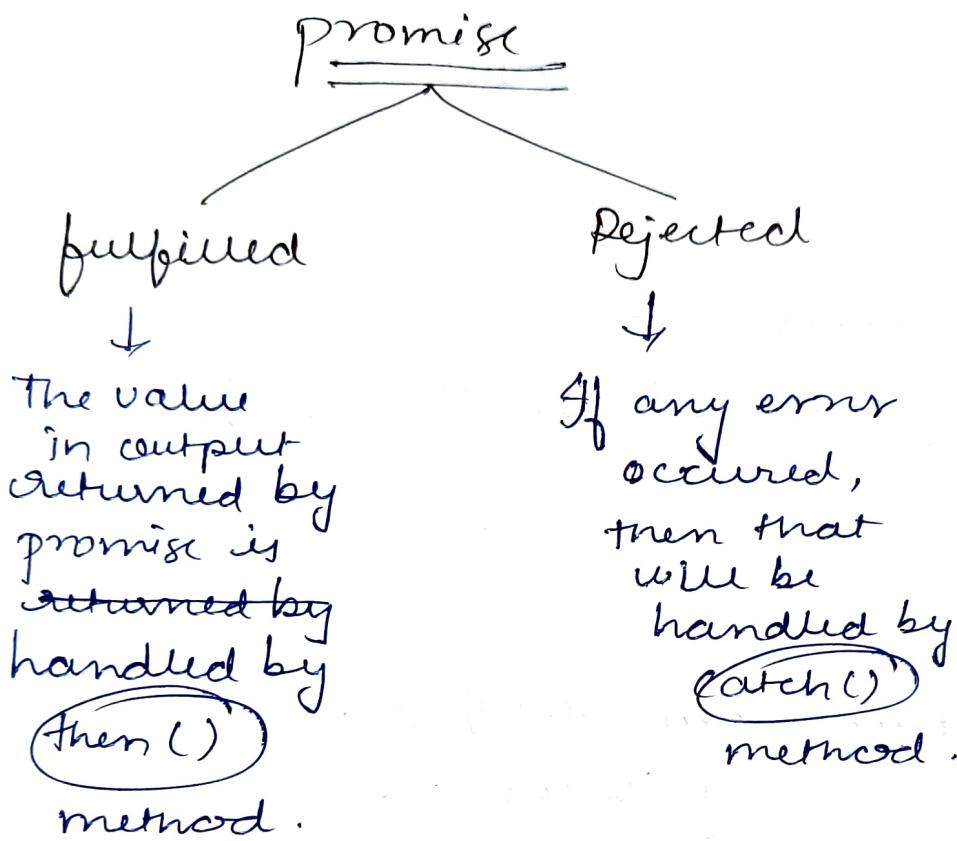
\downarrow

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 **then()** & **catch()**

```
graph TD; then --- value_desc; catch --- error_desc;
```

then()

For value or resolve.

catch()

For error or reject

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

```
  setTimeout(function() {  
    console.log('I am inside promise');  
  }, 5000);
```

```
// resolve(12345);
```

```
// reject(new Error('Error'))
```

```
});
```

) myPromise.then((value) => {

```
  console.log(value);
```

↓
will give 12345 output.

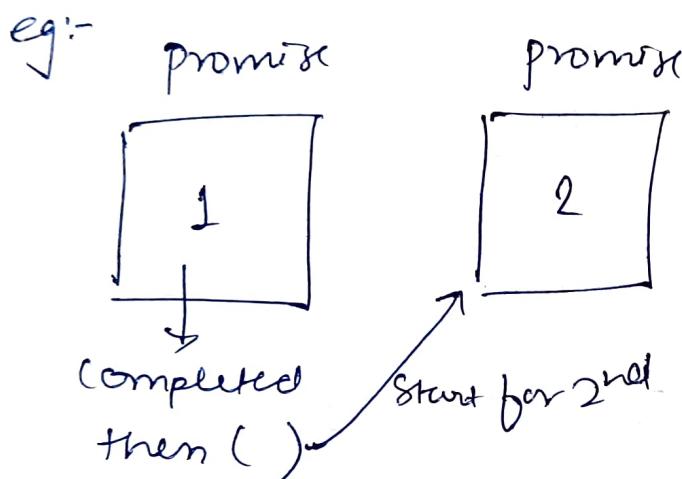
) myPromise.catch((error) => {

```
  console.log(error);
```

↓

will give 'error' output
written after
reject error occurred

- ✓ We Don't let our Synchronous Code wait for Asynchronous, we let Asynchronous work in background parallelly, 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:-

```

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("waada2 resolved");
    }
  )
  
```

here one more promise with 3000

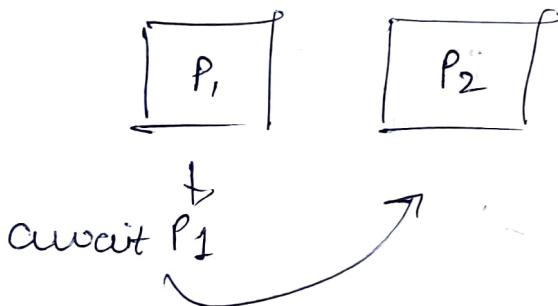
return value;

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

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



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

————— x ————— x —————
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 wait else
make it run
they will 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)

Fetch 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 (usuid), 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'

 headers :

:

:

}

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();
```

↑
an object is
sent in url
to update
data.

headers is additional
information

JSON.stringify()

↳ converting object notation to String.

Format conversion ✓

New,

Closures :-

creating function inside function

function abcd ()

 var name = "xyz";

 function displayName()

 console.log(name);

 displayName();

~~start~~; abcd();

xyz printed

✓ let is a block Scrope

if we will use let in place of var then also xyz will be printed

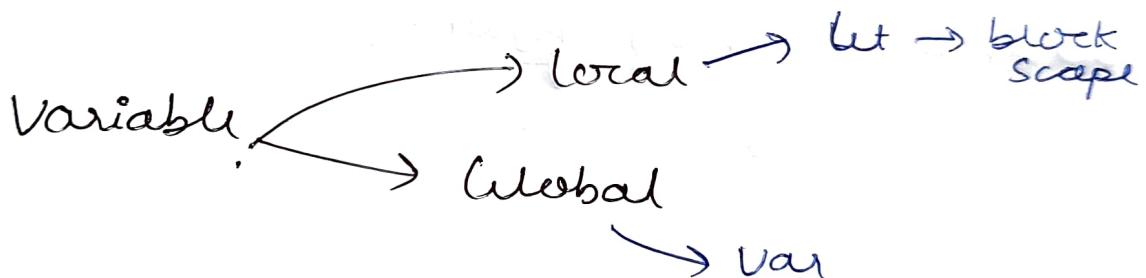
```

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

```

↳
init();

↳ Babbar will be
printed



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

if you will call

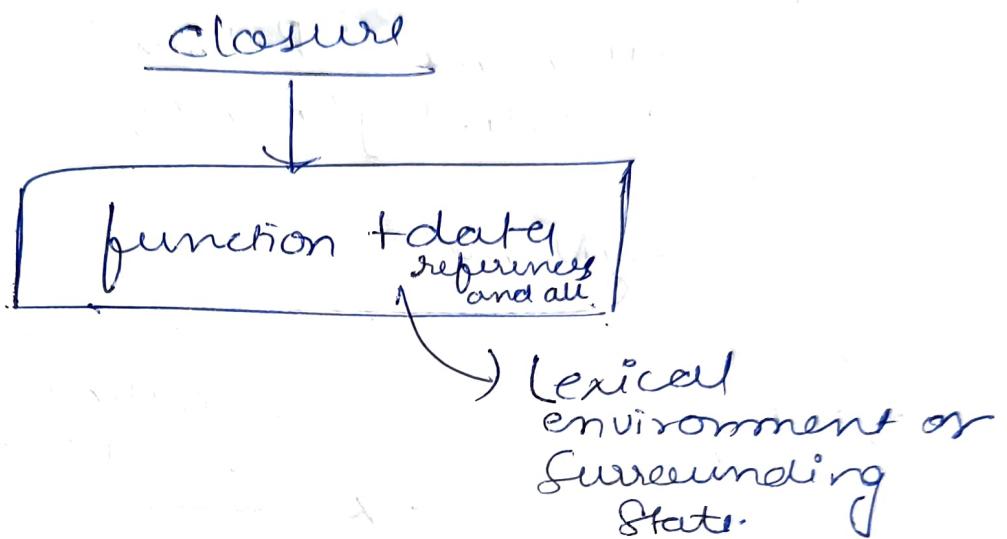
let funct = init();

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

X

X