Javascript

<head> </head>
<body>
<script> </script> // <script src="first.js"> </script>
</body>

if JS code is written blw <script tag>

if JS code is written in external file

script tag is written usually before </body>

COMMENTS IN JS

- // - single line comment
- /* */ - multiple line comment

Operators in JS

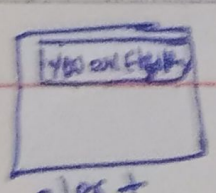
1 - Arithmetic +, -, /, %, *, ** (2**3 = 2^3 = 8)
2 - unary a++, ++a, a--, --a
3 - Assignment =, +=, -=, *=, %=, **=, /=
4 - Comparison ==, != (Value), ===, !== (Value with datatype), >, >=, <, <=

a = 5, b = "5"
console.log(a==b) // True
console.log(a===b) // False

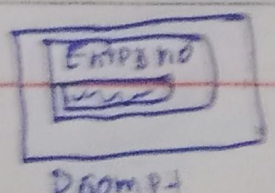
5 - logical &&, ||, !

Printing in JS

- console.log(5); // in console 5 will be Print in browser
- alert("You are Eligible"); // in browser dialog box popup
- prompt("Enter no."); // in browser dialog box popup with input



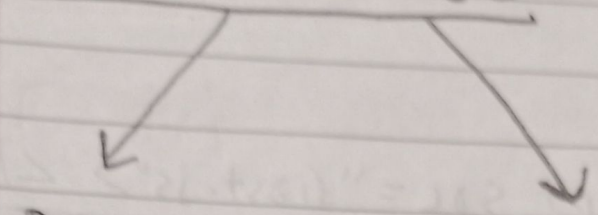
alert



Prompt

Date :
Page No. 2

DATA TYPE IN JS



	Primitive(7)	Non-Primitive.
1	Number	+ object
2	String	
3	Boolean	
4	undefined	
5	NULL (It is object but consider primitive datatype)	
6	Big int	
7	Symbol.	

Variable RULE

- Case sensitive $a \neq A$
- only letter, digit, underscore (_), $ is allowed (space not allowed)
- first character → letter / underscore / $ (number not allowed).
- Reserved word can't be variable (reserved console not allowed but Console allowed)
  Capital C not reserved.

SCOPE OF VARIABLE

(global) var : global scope variable, can be redeclared, can be updated
(block) let : block scope variable, can be update d
(fixed) const : block scope variable (its value can't changed)

	var a = 5	let a = 4	const a = 5
redeclared	var a = 6		
updated	a = 7	a = 8	

TYPECASTING IN JS

```
Var a = Symbol (5345);
console.log (a)  // Symbol (5345)

Var a = String (1234);
Console.log (a)  // '1234'

Var a = BigInt (1234);
console.log (a)  // 1234n  (n indicate bigInt)
```

typeof(a) function

used to know datatype

```
console.log(typeof (a))  // symbol

                         // string

                         // BigInt
```

CONDITIONAL STATEMENT

- esle-if

```
If (condition) {     }
elseIf (condition) {   }
else {     }
```

- ternary operator

```
let a = 25>18 ? "adult" : "not adult";
        condition.  true condition  false condition
```

- Switch

```
Switch (condition) {
case1:          ;
break;
case2:          ;
break;
default:        ;
}
```

LOOPS IN JS

- For loop

```
for (let i=1; i<=5; i++){

}
```

- while loop

```
while (condition) {

}
```

- do while

```
do{

}while (condition);
```

- For in loop
(index)

```
var a=[8,5,7,34];

for (var x in a) {
    console.log(x); //0123
}                   (index)

for (var x in a){
    console.log(a[x]); //85734
}
```

- for of loop : used for iterating array, string etc

```
var a=[8,5,7,34];

for (var x of a){
    console.log(x); //85734
}
```

Date :
Page No. 5

String in JS (It is immutable like Java)

```
Var str = 'Abhi' ;
Var str2= "Abhi" ;
          0123
```

⇒ console.log(str.length); // 4 // LENGTH OF STRING
⇒ console.log(str[3]); // i // CHARACTER AT PARTICULAR INDEX

⇒
```
Var str3= "Ab\nni" ;   // \n escape character
          01 234
console.log(str3.length); // 5 (\n is also count as 1)
console.log(str3[2]); // nothing will be displayed as \n
cosole.log(str3); // Ab
                  //  hi   (\n due to this)
```

METHODS IN JS FOR STRING

- str.toUpperCase( ) — ABHI
- str.toLowerCase( ) — abhi
- str.trim( ) — remove spaces from left and right (" Abhi " → "Abhi")
- str.slice(1,3) — bh (1 included and 3 excluded) (if it is missing then whole string from 1 to end)
- str.concat(str3, str2); — AbhiAb
- str.replace('bh','23') — A23a3li
- str.charAt(2) — b

hi Abhi

* str.slice works with negative index too like in array

Q.
```
let item = 'pen' ;
let cost = 10;
console.log("the price of ", item, "is " cost);
// the price of pen is 10
```
⇒ console.log(`the price of ${item} is ${cost}`); // Template literal
// the price of pen is 10

Arrays in JS (collection of items.

```
let info = ["rahul", 86, "Delhi"] // mixed not prefer
let marks = [56, 78, 91, 84, 73] // number type.
let heroes = ["ironman", 'hulk'] // string type
```

- Print Array :- console.log(info); //["rahul", 86, "Delhi"]
- length of Array :- console.log(info.length); // 3
- type of Array :- console.log(typeof(info)); // object
- index of Array :- console.log(info[1]); // 86

ARRAY METHODS IN JS

- push(), pop() at last - unshift(), shift() at first

```
marks.push(33); // 56,78,91,84,73,33
marks.pop(); // 56,78,91,84,73
marks.unshift(5); // 5,56,78,91,84,73
marks.shift(); // 56,78,91,84,73
```

- tostring : marks.toString(); // array become string

- concat : marks.concat(heroes,info);
  // [56,78,91,84,73,"ironman",'hulk',"rahul",86,"Delhi"]

- slice (similar to string's slice)

marks = [56, 78, 91, 84, 73]
+ve index (0, 1, 2, 3, 4)
-ve index (-5, -4, -3, -2, -1)

+ve index:
```
console.log(marks.slice(1)); // 78,91,84,73
console.log(marks.slice(1,3)); // 78,91
```
(3 excluded.)

-ve index:
```
console.log(marks.slice(-2)); // 84,73
console.log(marks.slice(-4,-2)); // 78,91
```
(-2 excluded)

- Splice (index, no-of next element to be deleted, insert data).

```
let x = [6,8,4,6];
x.splice(2,0,"abhi","yumi","mani");
console.log(x); // 6,8,"abhi","yumi","mani",4,6
```

(2 → 0, 1, 2, 3)

```
let x = [6,8,4,6]
x.splice(2,1,"abhi");
console.log(x); // 6,8,"abhi",6
```

(4 is deleted ① element.)

```
let x = [6,8,4,6]
x.splice(2,2,"abhi");
console.log(x); 6,8,"abhi"
```

(4, 6 is deleted ② element)

OBJECT IN JS (Similar to Hashmap in Java and associative array in PhP)

```
int x = {
name: 'abhishek'
cgpa: 9
i: 8989
}
```

Analogy (key : value) Analogy (arr[key] = value)

Accessing object

```
console.log(x) // { name: 'abhishek', cgpa: 9, i: 8989 }
console.log(x.name) // abhishek
console.log(x["name"]) // abhishek  ("literal is used as name is string and used as index)
console.log(x[1]) // 8989  ("literal not necessary as 1 is number and used as index)
console.log(x['1']) // 8989
```

Date :
Page No. 8

FUNCTION

non-parameter
```
function greet() {
    console.log("Radhe Radhe");
}

greet(); // Radhe Radhe
```

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

console.log(sum(3,4)); // 7
```

```
typeof(greet) // function
typeof(greet()) // it will run & return
```

ARROW FUNCTION

↗ compact way of writing a function

```
var greet = () => {
    console.log("Radhe Radhe");
}

greet(); // Radhe Radhe

var sum = (a, b) => {
    return a+b;
}

console.log(sum(3,4)); // 7
```

ForEach , filter, map, reduce

return new array with logic — return new array — reduce to single value and return number

```
var x = [4,5,7,3,8]

x.forEach(
(value, index, array) => {
    console.log(value, index, array);
}
);
```

value	index	array
4	0	[4,5,7,3,8]
5	1	[4,5,7,3,8]
7	2	[4,5,7,3,8]
3	3	[4,5,7,3,8]
8	4	[4,5,7,3,8]

storing because map return new array.

```
var y = x.map(
(value, index, array) => {
    if (index % 2 == 0) {
        return value ** 2;
    }
    return value;
}
);
console.log(y) // 16, 5, 49, 3, 64
```

we store in z → because filter return new array

```
Var z = x.filter (
    (value, index, array) => {
        return value%.2 == 0;
    }
)
console.log (z); // [4, 8]
```

NOTE :- value, index, array are just variable name we can use anything in all method, forEach, map, filter and reduce

```
x = [4, 5, 7, 3, 8]
```

we store in a → as reduce will return number

```
Var a = x.reduce (
    (accumulator, value, index, array) => {
        return accumulator * value;
    }, 3,
);
console.log (a); // 10080
```

③ It is accumulator's initial value If we don't define it then first element of array i.e 4 will become accumulator and next process will goon

clear way to writing forEach, map, filter and reduce

```
Var x = [4, 5, 7, 3, 8]
```

Arrow

- Using function

```
Var s = (value, index, array) => {
    console.log (value, index, array);
}

x.forEach (s);
```

(s() x sv)

- Using ~~Arrow~~ function

```
function s (value, index, array) {
    console.log (value, index, array);
}
x.forEach (s);
```

(s() x sv)

☆ similarly, map, filter, reduce work

Objects and Prototypes

concept - 1 : object

```
let y = {
  hero(){
  console.log("You are hero ji", x.name);
  }
}

let x = {
   name : "abhishek",
   p : y.hero
}

x.p(); // You are hero ji abhishek
```

(y.hero() X)

concept - 2 : Prototypes

```
let y = {
  hero(){
  console.log("You are hero ji", x.name);
  }
}

let x = {
   name : "abhishek"
}

x.__proto__ = y
```

(function of y can be used by x)

```
x.hero(); // You are hero ji abhishek.
```

OOPS IN JS

Class is of function type

```
class Animal {
    animal_name;                    // Variable

    constructor (name){             // constructor.
        this.animal_name = name;
    }

    speak(){                        // method
        console.log(`${this.animal_name} makes a noise.`);
    }

    isAnimal() {
        console.log("Yes, it is an Animal.");
    }
}

class cat extends Animal {          // inheritance
    cat_name;
    #price;                         // Private variable

    constructor (name, price){
        super(name);                // calling parent class's constructor
        this.cat_name = name;
        this.#price = price;
    }

    speak() { console.log(`${this.cat_name} meows!.`);
    }
```

```
#getPrice() {                    // Private method to get Price
    return this.#price;
}

Showprice() {
  consolelog(`The price of ${this.cat_name} is
                    $${this.#getPrice()}`);
  }                 $100

}

Creating object

const mycat = new Cat("whiskers", 100);

mycat.speak(); // "whiskers meows!"
mycat.isAnimal(); // "yes, it is an animal"
mycat.showPrice(); // "The price of whiskers is $100."
console.log(mycat.cat_name); // "whiskers"

console.log(mycat.#getPrice()); // error
console.log(mycat.#price); // error

consolelog(typeof(Animal)); // function
console.log(typeof(Cat)); // function
```

can be var and let but const prefer be used

ERROR HANDLING IN JS

If error occur in program at particular line then below that line all programs code's stopped.

```
console.log("Hello");
console.log("Namaste");
console.logs("Paji");      → error occur here
console.log("Praji");      } all this code will not work. below error line. to overcome this try and catch and finally block is used.
console.log(4);            }
```

output

Hello

Namaste

error.

type of error.name = RangeError, SyntaxError, TypeError, Error } all are calls sent to the

Date :
Page No. 14

```
let age = 148;
function agecheck(){
  try {
  if(age>100){
    throw new RangeError("age jada hai")
          // error.name(error.message)
  }
  if(age<0){
    throw 'negative number'
          // error
  }
  if (age==18){
     throw 18;
     // error
  }

  console.log ('You can vote');
  }

  catch (error)   // if in try block error occurs we send to catch.
    console.log (error);     // RangeError: age jada hai
    console.log (error.name); // RangeError
    console.log (error.message); // age jada hai
  }

  finally {          // It will definitely work even error occur
    console.log('End');      in catch block due to any typing mistake like,
  }                          console.logs('Hello');
}

agecheck();
console.log ("Next line code"); // if error occurs in catch block
                                   then this line and below this line of code
                                   won't work
```

Way-1

Way-2

Way-3

error is just variable we can use anything like e, fac, anime etc

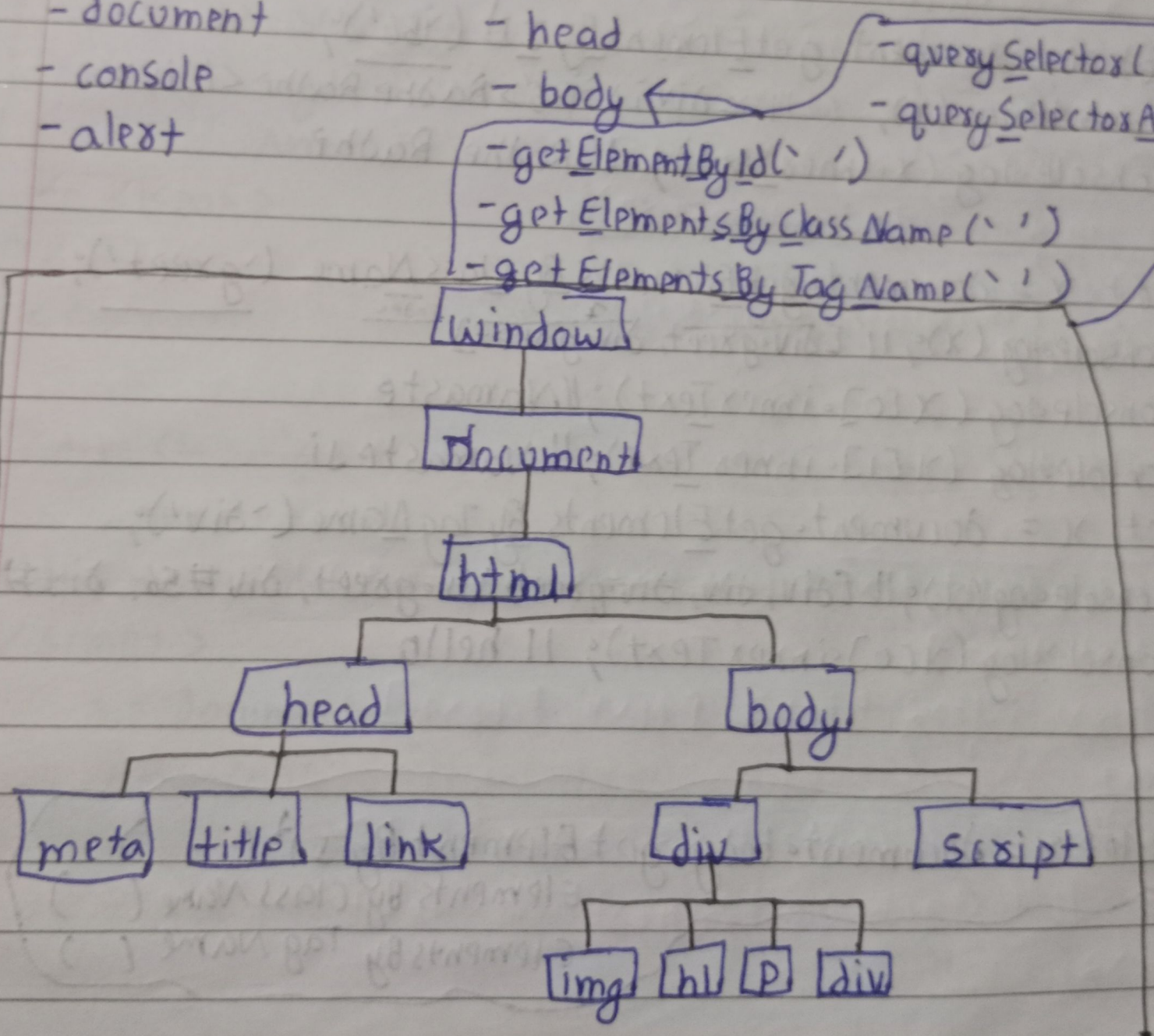
DOM Manipulation

When page is loaded, DOM (Document object model) is created by browser.

Window object :- Represent an open window a browser.
It is browser's object (not JS's) & is automatically created by browser
- It is a global object with lots of properties and methods

Window
- document
- console
- alert

document
- head
- body
- querySelector(' ')
- querySelectorAll(' ')
- getElementById(' ')
- getElementsByClassName(' ')
- getElementsByTagName(' ')



~~Ex- console.log(window.console.log(2));~~

Ex- window.console.log(2); // console.log(2); } both has same output as window is global

DOM MANIPULATION

```
<div> hello </div>
<div> hello2 </div>
<div class = "greet"> Namaste </div>
<div class = "greet"> Namaste ji </div>
<div id = "i56"> Radhe Radhe </div>
<div id = "i56"> Good morning </div>
```

it will run but two tag should never have same id because while fetching id first one fetched only. i.e Radhe Radhe

* id should start with alphabet and must be unique

return single tag as id is unique

```
let x = document.getElementById ('i56');
console.log (x);   // <div id = 'i56'> Radhe Radhe </div>
console.log (x.innerText);  // Radhe Radhe
```

It return array as many tag can have same class name

```
let x = document.getElementsByClassName ('greet');
console.log (x);  // [div.greet, div.greet]   Index
console.log (x[0].innerText);  // Namaste
console.log (x[1].innerText);  // Namaste ji
```

It return array as same tag can be used like above whole code written in div tag.

```
let x = document.getElementsByTagName ('div');
console.log (x);  // [div, div, div.greet, div.greet, div#56, div#56]
console.log (x[0].innerText);  // hello
```

```
let x = document.body.getElementById ( )
                     ElementsByClassName ( )
                     ElementsByTagName ( )
```

Query Selector

It return first element tag

```
let x = document.querySelector ('#i56') // ('.greet') // ('div')
```

It return array of same tag, id and class name

```
let x = document.querySelectorAll ('#i56') // ('.greet') // ('div')
```

Date :
Page No. 17

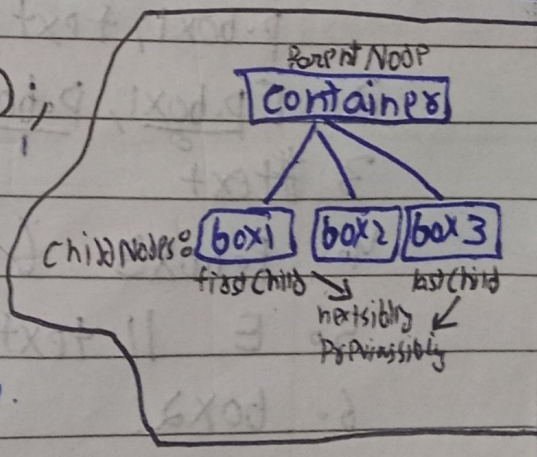
After selecting tag : Properties

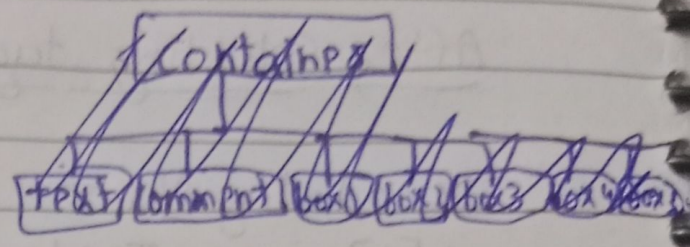
x[0].tagName :- return tag name
x[0].innerText :- return text content
x[0].innerHTML :- return html
x[0].textContent :- return text content for hidden element

```
<div class="grept">
    Hi
    <p>
        goodmorning
        <span style=" visibility: hidden;">
            Namaste
        </span>
    </p>
</div>

<div class="grept">
    Namaste
</div>

<script>
let x = document.querySelectorAll(".grept");
• console.log(x);            // [div.grept, div.grept]
• console.log(x[0].tagName);  // DIV
• console.log(x[0].innerText); // Hi
                               goodmorning.

• console.log(x[0].textContent); // Hi
                                 goodmorning
                                 Namaste (hidden but still display in console)

• console.log(x[0].innerHTML); // Hi
                               <p> goodmorning.
                                   <span style="visibility: hidden;">
                                       Namaste </span>

</script>
```

Type of Nodes.
- text — hello
- comment — //hello
- element — hello

text, comment, element	Element
childNodes (it will return array)	children (it will return array)
firstChild	firstElementChild
lastChild	lastElementChild
previousSibling	previousElementSibling
nextSibling	nextElementSibling
parentNode	parentElement






```
<div class="containes">
  <!--boxes--> E
 <P class="box1"> box1 </P>a
 <P class="box2"> box2 </P> b
 <P class="box3"> box3 </P> c
 <P class="box4"> box4 </P> d
 <P class="box5"> box5 </P> e
 </div>

 <script>
   x = document.body.querySelectos('div');
 1 console.log (x.childNodes) // [text, comment, text, p.box1, p.box2, p.box3,
 2 console.log (x.children) //
 3 console.log (x.childNodes[2]) //
 4 console.log (x.children[2]) //
 5. console.log (x.childNodes.textContent)
 6. console.log (x.children[1].innesText);
 </script>

 //output

 1 [text, comment, text, p.box1, text, p.box2, text, p.box3, text, p.box4, text,
   p.box1, text]
 2 [p.box1, p.box2, p.box3, p.box4, p.box1]
 3 #text
 4. p.box3 ( <p class="box3"> box3 </p>
 5. E // text node don't support innes text
 6. box2.

 x = document.body.queryselectos ('.box2');
 console.log (x.nextElementSibling) // <p class="box3"> box3 </p>
 console.log (x.previousElementsibliy) // <p class="box1"> box1 </p>
```

Dom manipulation

 Hello

Get Attribute

```
x = document.body.querySelector('div');
console.log(x.getAttribute('class')); // container
Console.log(x.getAttribute('id')); // d123
```

if exists that return value else null

fixed class and id we can't use other variable

SET attribute

```
x.setAttribute('class', 'con');
x.setAttribute('id', 'd12');
//output
<div class='con' id='d12'> Hello </div>
```

If exist update value else add that attribute if id doesn't exist and we set id then id will be added to that tag

Styling

```
x.style.backgroundColor = 'red';
x.style.fontSize = '32px';
```

CSS syntax	CSS of JS syntax
background-color	backgroundColor
font-size	fontSize (camelCase)
border-radius	borderRadius
color	color

deleting tag/node

```
x.remove(); // It will delete that tag/node from website
```

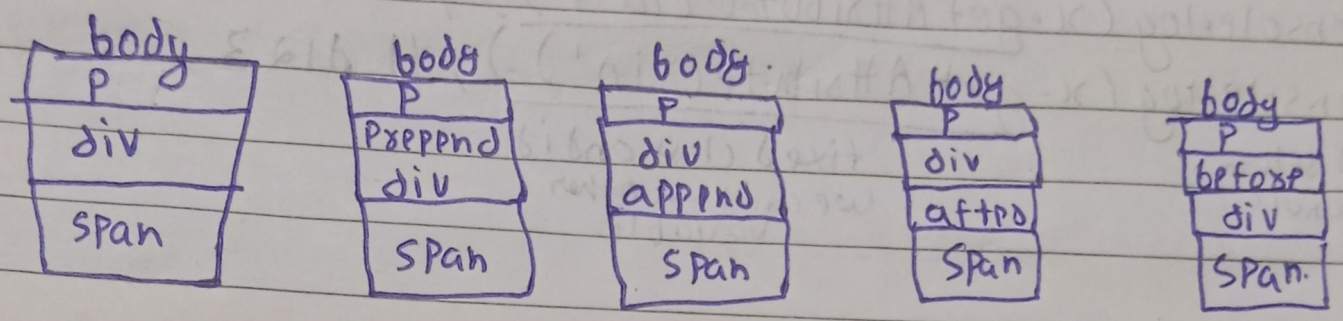
creating element/ tag/node

```
let nod = document.createElement('div');
nod.innerText = 'hello world';
nod.setAttribute('id', 'd234');
nod.setAttribute('class', 'cj3');
```

```
//output
<div class='cj3' id='d234'> hello world </div>
```

adding created Element/ tag/ node to div.

- div.prepend(nod) → at top
- div.append(nod) → at bottom
- } inside tag
- div.before(nod)
- div.after(nod) → at bottom
- } outside tag.



Event and Event listeners

Event Handlers

way-1 click me

way-2 click me

<script>
function mes(){
console.log('button was liked');
}

~~Key~~ Event Pointer	onclick - one one click of mouse ondblclick - one double click of mouse
mouse Event	onmouseover - when hover onmouseout - when moving out after hovers
Keyboard event	onkeydown - first key go down onKeypress - second key get pressed/Holding Pr onkey up - thirdly when we leave button key go up

way-3 click

<script>
let x = document.body.querySelector(".g");

x.onclick=()=>{
console.log("button was click");
}

x.onclick = function() {
console.log("button was click");
}

</script>

way-4.

function mes(){
console.log("button was click");
}

x.onclick = mes; // if we use x.onclick = mes() then function run after loading page and don't wait for onclick

Date :
Page No. 22

Event object

```
x.onclick = (e) => {
    console.log (e)      // e is event object of array type.
}                        we can use any other name of e like event, adam etc
```

e.type - return type of event → click, dblclick, mouseover, mouseout, keypress, keydown, keyup

e.key - return value entered by keyboard event.

e.clientX / clientY - return location at x-axis and y-axis of Mouse event and pointer Event.

e.target - return tag

e - MouseEvent : type target clientX clientY ~~key~~
PointerEvent : type target clientX clientY ~~key~~
KeyboardEvent : type target. ~~clientX~~ ~~clientY~~ key

Event listeners (It is used when we want to add more than one event)

```
x.onclick = () => {
  console.log("clicked 1");
}
x.onclick = () => {
  console.log("clicked 2");
}
```

this will overwrite above one

// output on click : clicked 2.

Same event use cause overwriting but we can overcome this problem using event listeners

but if we use event listeners then we can add as many as same event on Element without overwriting it

click
dblclick
mouseover
mouseout
keydown
keypress
key up

adding event listen

```
const button = document.querySelector('.g');
button.addEventListener('click', b=()=> {
    console.log("Button was clicked");
});

button.addEventListener('click', b=()=> {
    console.log("second event listeners for click");
});
```

// output
Button was clicked
second event listeners for click

remo ving

```
button.removeEventListeners('click', c);
// output
```

here only function name is given we can't write function here because inspite of same code of function define in different

Asynchronous Coding

Date :
Page No. 24

Synchronous code

```
console.log("Radhe Radhe");
console.log("Namaste");
console.log("Good Morning");
console.log("Hello");

// output
Radhe Radhe
Namaste
Good Morning
Hello
```

Asynchronous code : need in time of API to execute other line of code until we get data from API which is taking time.

```
console.log("Radhe Radhe");
console.log("Namaste");
setTimeout(()=>{console.log("Bye")}, 4000); // 4000=4s
console.log("Hello");

// Output
Radhe Radhe
Namaste
Hello
Bye
```

asynchronous function

`setTimeOut(function, timeinms)`

← (It execute but take 4s to run function meanwhile other line of code execute i.e Hello)

Callback function:- Callback is a function passed as an argument to another function.

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

function — Callback function passed as an argument

```
function calc(a,b,s){
    return s(a,b);
}
```

```
console.log(calc(2,3,sum)); // 5
```

Callback Hell:- difficult to understand

callback function.

```
function getdata(id, nextid){
    setTimeout(()=>{
        console.log("data is", id);
        if(nextid){          // in last we don't pass nextid ∴ we use this condition
            nextid();  // chaining call
        }
    }, 2000);
}
```

```
getdata(1, ()=>{      // getdata(1) is passed with nextid in getdata(2) function
    getdata(2);       // getdata(2) is passed without nextid.
});
```

```
getdata(1);
getdata(2);
```

— to ignore this we pass nextid function in getdata(id) to getdata(id, nextid);

```
//output:-
1  (At 2sec)
2  (At 2sec)   } immediate
```

```
//output:-
1  (At 2sec)   }
2  (After 4sec) } one after other at sleep of 2
```

PROMISE

Promise : better than callback hell.

It is an object in JS. Promise is for "eventual" completion of task.

API send vs Promise.

let promise = new Promise((resolve, reject) => { })

function Handler created by JS by itself

Promise has 3 state
- Pending
- fulfilled (resolved)
- Rejected

function send message. Can use different name.

```
let P = new Promise ((resolve, reject) => {
    console.log ("hello");
});
console.log (P);
```

let P = new Promise (function)
resolve ("success 331")
reject ("Error mil gayi")

Pending state

//output : hello
[[PromiseState]] : "pending"
[[PromiseResult]] : undefined

```
let P = new Promise ((resolve, reject) => {
    console.log ("hello");
    resolve ("success 333"); }); // reject ("gadbag omg")
console.log (P);
```

resolve() and reject() fulfilled and rejected state

//output : hello.
[[PromiseState]] : "fulfilled" (rejected)
[[PromiseResult]] : "success 333" / gadbag omg.

if promise return resolve
.then(function) & .catch(function)
if promise return reject

then for resolve

```
function api(){
  return new Promise((res, rej)) => {
    console.log("hello");
    res("success 3331");
  })
}
```

```
p.then(function)
p.catch(function)
    function          function
    (res) => {        (rej) => {
        }                 }
```

Way-1
```
let p=api();
p.then((res) => {
    console.log(res);
})
```
output: hello
success 3331

Way-2
```
api().then((res) => {
    console.log(res);
})
```
output: hello
success 3331

catch for rejet

```
function api(){
  return new Promise((res, rej)) => {
    console.log("hello");
    res("error sorry");
  })
}
```

Way-1
```
let p=api();
p.catch((rejet) => {
    console.log(rejet);
})
```
output: hello
error sorry

Way-2
```
api.catch((rejet) => {
    console.log(rejet);
})
```
output: hello
error sorry.

☆ If api return resolve then we can use then(function) if we use catch(function) then it won't work and vice-versa.

PROMISE CHAINING

```
function getdata(id){
  return new Promise ((resolve, reject) => {
    setTimeout(()=>{
      console.log ("data", id);
      resolve("success");
  }, 3000);
});
}

getdata(1)
.then((res) => { return getData(2); })
.then(res) => { return getdata(3); })
.then((res) => { console.log(" Programme End"); });
```

used for chaining.

//output

data 1 at 3rd sec
data 2 at 6th sec
data 3 at 9th sec
Programme End at 9th sec

Async- Await

async function always returns a promise
await pause the execution of its surrounding async function until the promise is settled.

```
function greet() {
 console.log ('Hello');
 }
```

```
async function greet() {
   console.log ("Hello');
   }
```

We use await function inside async function ~~until~~ else error come

If we use async- Await then we don't need to use .then and .catch method but we use promise.

Ex -

```
function api(id){
  return new Promise ((resolve, reject)=>{
      setTimeout(()=>{
        console.log ("hello", id);
        resolve("Success 3331");
      }, 3000);
  })
)

async function getdata(){
   await api(1);
   await api(2);
   await api(3);
} getdata();
```

// output hello 1 at 3rd sec
hello 2 at 6th sec
hello 3 at 9th sec

IIFE : Immediately Invoked function Expression

IIFF function is called immediately as soon as it is defined.

(fn)()();

- (function(){})()
  f(x)

- (()=>{})()();
  f(x)

- (async ()=>{})()();
  f(x)

Ex -

```
function greet(){
    console.log("Radhe Radhe");
}
greet(); // After calling we get output i.p Radhe Radhe.

(function(){
    console.log("Radhe Radhe");
})();
```

// immediately run and we get output i.p Radhe Radhe without calling function.

API (Application Programming Interface)

fetch API provides an interface for fetching resources.
It uses Request and Response objects.
fetch() method is used to fetch a resources/data.

let p = fetch(url, [options]) // p is promise

json() method: input is json and output is js object

HTTP response status code.	
100 - 199	Informational responses
200 - 299	Successful responses
300 - 399	Redirection responses
400 - 499	client error responses
500 - 599	server error responses

```
let p = await fetch(url);
  fetch(url)
response.json
```

```
(async function(){
let url = 'https://cat-fact.com/facts';
let response = await fetch(url);
let data = await response.json();
console.log(data[0].text);
})();
```

depend upon API

API fetch → data
Promise
using Async-await

```
(function(){
let url = 'https://cat-com/facts';
fetch(url)
.then((response)=>{
    return response.json();
})
.then((data)=>{
    console.log(data[1].text); });})();
```

API fetch → data
Promise
using .then and .catch