Javascript

HTML + CSS + Javascript
↳ Add functionality

REPL
- Read
- Evaluate
- Print
- Loop.

console window
clear → ctrl+l

Variable → name of a storage location

console → a = 10
b = 20
name = " "

Primitive Datatypes
- Number (10, 20)
- Boolean
- Undefined
- String
- Null
- Bigint

typeof 5.9
number
5.6
+9 -1 2 0

a = 0.999999999999999
↳ 1

NaN → Not-a-number

type of "hello"
↳ string.

0/0 → NaN

typeof NaN
↳ number

NaN + 1 → NaN
NaN * 2 → NaN
NaN * NaN → NaN

Priority: (+,-) (*,/,%) , (**) , ( ) — increasing
4 3 2 1

→ let or var
syntax of declaring variables

let a = 6;
let b;

→ const
↳ values of constants can't be changed with reassignment & they can't be re-declared.

RANKA
DATE / /
PAGE

Identifier rules → unique names given to variables

- Names can contain letter, digits, underscores and dollersigns (no space)
- Names must begin with a letter
- Names can begin with a $ and _
- Names are case-sensitive
- Reserved keywords cannot be used as names.

```
isAdult = true;   → no quotes.
a = true;
typeof a
↳ Boolean
```

Typescript → Static Typed, not dynamic, designed by M

JS is dynamic typed
↳ variables datatype can be changed.

Strings

null in JS variables → not been assigned
↳ intentional absence of any object value to be explicitly assigned.

```
let a = null;
```

undefined → a variable that has not been assigned a value is of type undefined.

```
let a;
a
undefined
```

String Interpolation → Back ticks

`String Text ${ expression } string text`

RANKA
DATE / /
PAGE

console.log() → write msg.

```
k = 9;
console.log("Apna", "College", 2, k);
```

JS browser
console.log()

→ Writing JS File index.html, app.js

```
<script src="app.js"></script>
</body>
```

61 41
'a' < 'A' → false

ASCII characters are associated to unicode

```
let a = 5;
let b = 6;
console.log("sum is:", a+b);
```

Template literals ${a+b}

```
let output = `The output is: ${b}`;
```

all expressions to be embedded into strings

lowercase
'a' < 'b' < 'c' < 'd' ..
'A' < 'B' < 'C' < 'D'

n == str → type not compared, only value compared

n === str → compares value and also compares type.

Truthy & Falsy values →

Everything in JS is true or false (in boolean context)

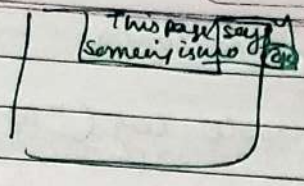
falsy → false; 0; -0; 0n (BigInt value); "" (empty string)
null, undefined, NaN

Truthy → everything else

if("abc") → true
if("") → false
if(null) → false
if(" ") → true.

RANKA
DATE / /
PAGE

alert ("something is wrong");

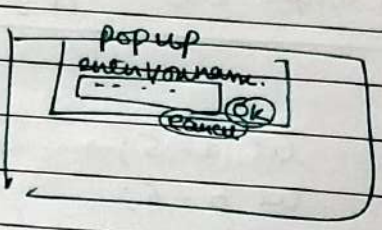


console.error ("error");

⊗ error

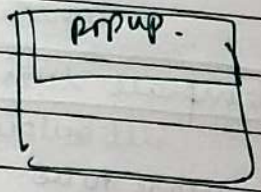
console.warn ("warning");

Prompt



prompt ("Enter your name");

let name = prompt ("enter name");



String methods

Str.trim()
↳ remove extra spaces (trailing)
starting & ending

Strings in JS are immutable.

Str.toUpperCase()
Str.toLowerCase()

I love coding
01

Str.IndexOf ("love")
↳ first index ⇒ 1

method chaining
one after other
in methods

Str.trim().toUpperCase()

Str.slice (5) = coding

I love coding
0 1 2 3 4 5

Str.slice (1, 4) = love
to
end index non inclusive

Str.slice (-num)
= Str.slice (len-num)

Str.slice (len-num)

Str.substring(——)

Replace → searches & then replaces

Str.replace("love", "do") Idocoding.

replace with

Repeat ⇒ returns a string with the no. of copies

msg.repeat(2)

2 times

msg -

msg -

Arrays →

let students = ["aman", "ram", "karan"];

let info = ["aman", 45, 6.0]; //mixed array.

let newArr = [];

arr.length;

Arrays are mutable

value

Array methods → push() → add at last

pop() → delete from end & return

unshift() → add to start

Shift() → delete from start & return

concat

[arr.concat(arr2)]

[arr.reverse();]

indexOf("value") → gives index of value in array

not found → -1

includes("value") → true → found in array

↳ not found → false

RANKA
DATE / /
PAGE

slice in array → copies a portion of an array
returns copy

colors.slice() → returns original array copy

[a, b, c]
-2 →

color.slice(2) → 2 to end values.

colors.slice(2, 5) → 2 to 4 all values picked

colors.slice(-2) → end to -2
last second elements

Splice : remove, replace, add elements in place

splice(start idx, delete count, items to be added)
(optional)

colors.splice(4)
acts as slice

colors.splice(0, 1)
start from 0 delete 1 element : delete 0th element

colors.splice(0, 1, "black", "grey")
deleted → replace

0 1
["black", "grey", ...]
0th

arr.sort()

chars & strings, int

arrays reference:

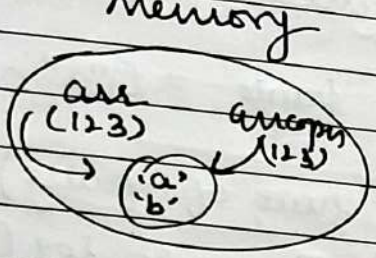
every time new array is created it gets a new memory and the values are accessed using variable name.

[1] === [1] false
[1] == [1] false

neither value is same nor address.

let arr = ['a', 'b'];
let arrcopy = arr;

changes in arr will be reflected in arrcopy & vice versa



arr == arrcopy
→ true

arr === arrcopy
→ true

const arr = [1, 2, 3, 4]

const pi = 3.14; pi = 8; → error.

→ arr.push(4); → no error → changes also made.
→ arr.pop(); → no error. In arr fnc can be done on const errors
→ arr = [1, 2, 3] → error.

Nested array → multidimension arrays.

let nums = [[4,4], [3,6], [4,5]];

```
for (let i=1 ; i<=5 ; i++) {
        console.log(i) ; }
```

parseInt(prompt value)

```
while (i<=5) { ...
   ... }
```

similar to java.

```
console.log( i , fruits[i]);
```

for of loop → new in javascript

```
let fruits = ["a" , "b" , "c"]
for ( fruit of fruits)
    { console.log(fruit); }
```

```
for( list of heroes) {
  // console.log(list) ;
  for( name of list ){
      console.log (name) ; } }
```

```
<script src="app.js"></script>
```

JS object literals : →
keyed collections & complex entities

property ⟹ (key, value) pair

Student { name – "Ram" ; age – 23 ; marks – 944 }

Objects collection of properties

let delhi = {
city : "Metropolitan",
location : "North", }

const student = { id : "Student",
name : "Shyam"
age : 23 ;
skills : ["java", "web d"] ; }

Get values : → student["key"] ;
→ objname.key ;

let name="Ram";

in JS variable cannot be accessed using . operator

obj.name ✗ error

obj[name] ✓

JS automatically converts obj keys into strings.

student.city = "Mumbai" ;
↳ update existing key.

student.gender = "female"
↳ not present earlier
when this written automatically created gender key.

number can be converted to string or others

delete student.city
delete a key

RANKA

object of objects:

```
const class = {                    → object
    student1 : { age : 14; }
    karan : { age : 13; }
}.
```

console.log(class.karan.age) → 13

class.karan : → { age : 13; }.

array of objects

const array = [ ] → array.

```
{ name : "Aman"
  age : 14 }          array[0] = Aman

{ name : "Sumit"
  age : 13 }          array[1]
[ ].
```

constant obj
Math class obj.

Math.PI
Math.E

Math. → floor, ceil, random, abs.

Math.random()

(0 to 1) → excluded.
default value

(0 to 9) generate integers.

Generate random integers:

Generate → Math.floor(Math.random()*10)

1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ← we want (1 to 10) random nums

← Math.floor(Math.random()*10) + 1;

RANKA
DATE
PAGE

random no. between 1 and 100

→ Math.random() * 100

Math.floor(Math.random() * 100) + 1;

limit, distance.

random no. between 1 & 5

→ Math.floor(Math.random() * 5) + 1;

random no. between 21 & 25

Math.floor(Math.random() * 5) + 20;

Functions in JS:

function functionname() {
// work }

call → functionname();

Block scope
↓
let & const

lexical scope → Nested function

function functionname(name, age) {
.... }

no need to write datatype.

console.log(`my name is ${name} & age is ${age} years old`)

function fn() {
let x = 5;
let y = 6;
function fn2() {
console.log(y); }
fn2(); }

lexical scope

fn expression

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

anonymous function

a a b c d d e
↑
charAt

Higher Order Functions →
Take on or multiple func as argument

this keyword

```
const student = {
    name: "Shagun",
    age : 23,
    marks: 48,
    getInfo() {
        let total = age + marks   → error
        → let total = this.age + this.marks
        console.log(total);
    }
}
```

this.name }

by default this → window's object

Try & Catch

```
try {
    //risky code }
catch {
    console.log("error"); }
```

Arrow functions :

```
const sum = (a, b) => { console.log(a+b); }
```
(arguments)

}.

nameless functions.

```
const pow = (a,b) => {
    return a**b ; }
```

Implicit return in arrow functions:

```
const mul = (a,b) => (
    a*b );

const sum = (a,b) => (a+b);
```

Set Timeout (window object)

Callback → when a fn is passed into another fn as a parameter.

Set Timeout (function, timeout)
↳ executes only once after x ms.
ms → 1000ms = 1s

```
Set Timeout ( () => {
    console.log ("Apna College"); }, 4000 );
console.log ("Welcome to");
```

fn
timeout = 4s.
waits for 4000ms

program execution won't stop after 1 statement

op → Welcome to
(4s gap)
Apna College

SetInterval:

```
let id = setInterval (fn, timeout)
setInterval ( () => {console.log("hi"); }, 2000 });
```

// after every 2s continuously hi will be printed.

to stop the continuous execution. → clearInterval(id)

RANKA
DATE / /
PAGE

this with arrow →

Arrow
- lexical scope
- parent → call scope.

function
Scope → this → calling object

global scope

```
const student = {
    name: "aman",
    prop : this,        → scope - window obj.
    getName: function () {
        return this.name;
    }                   → class scope is object. Student
    getMarks: () => {   → arrow
        console.log(this);
        return this.marks;
    }
}
```

student.getName()
→ aman

student.getMarks()
→ undefined as marks not defined for window.

parent scope → window

```
info2 : function () {
    setTimeout (function () {
        console.log(this);
    }, 2000);
}
```

Refers to current object here it is window.

```
getInfo : function () {
    setTimeout (() => {console.log(this);}, 2000);
}
```

Student as arrow fn used ∴ it refers to parent function that belongs to student obj

(n) => (n * n); // → square of an array

print Hello 5 times after 2s each. →

```
let id = setInterval(() =>
        { console.log("Hello"); }, 2000);
// at 10000 ms it should stop
setTimeout(() => {
        clearInterval(id); }, 10000);
```

arr.forEach(function)

```
let arr = [1, 2, 3, 4, 5];
let print = function (el) {
        console.log(el); };
arr.forEach(print);
```

or.

```
arr.forEach(function(el) { console.log(el); });
```

filter: FILTER

```
let even = arr.filter((num) =>
        (num % 2 == 0));
```

will filter elements from arr on the basis of passed function and store them in a new arr.

map: creates a new arr after operating passed fn on the array

```
let newArr = arr.map(fn name);
```

same size as arr.

```
let num = [1, 2, 3, 4];
let double = num.map(function(el)
        { return el * 2; });
```

⇒ double → [2, 4, 6, 8]

RANKA
DATE / /
PAGE

EVERY : Returns true if every element of array agrees true for some function. Else returns false.

arr. every (some function definition or name)

[1,2,3,4]. every ((el) => (el%2 == 0));
→ false

[2,4]. every ((el) => (el%2 == 0));
true

Reduce →

arr. reduce (fn (accumulator, element));

[1,2,3,4]. reduce ((res, el) => (res + el));
res = result, el = element

Will run for every element in the array. Then return the result into the res and the same res is passed to next call.

[1,2,3,4]

1 → (res, 1) => (0+1)
res = 1

2 → (1, 2) => (2+1)
res = 3

3 → (3, 3) => (3+3)
res = 6

4 → (6, 4) => (10)
→ ans.

Finding max in array using reduce →.

```
let nums = [2, 3. . . .];
let result = nums.reduce( (res, val) =>
        { if ( val > res)
            {res = val;}
          else
            { return res; }
        } );
```

DEFAULT PARAMETERS :

Giving a default value to the argument.

```
function func( a,b =2) {
        // dosomething }
```

```
function sum(a, b = 3) {
        return a+b; }
Sum( 2) ;
```

a=2 , b = 3.

[a ⇒ 5]

sum(a=3, b)

Sum (1)

a=1, b=undefined

we prefer to default parameters and parameter

Spread : Expands an iterable into multiple values.

```
function func (...arr) {
        // do something }
console.log(..."hello");
```

Hello

let arr = [1, 2, 3, 4, 2, 4, 8] ...arr → 1, 2, 3, 4, 2, 4, 8

Math.min (...arr)

```
let odd = [1, 3, 5, 7];
let even = [2, 4, 6, 8];

let num = [... odd , ... even];
        [ 1, 3, 5, 7, 2, 4, 6, 8 ];
```

Spread with object literals

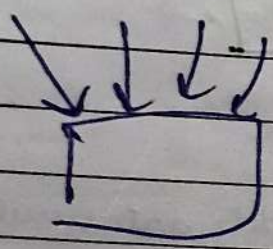
```
let data = {
    { email : "a..." ,
      password : "abc" } };

let dataCopy = { ... data , id: 123 }
```

```
let arr = [1,2,3,4]; //value
let obj1 = { ... arr }; //obj → key : value.
```

Index (key)	value (arr[i])
0	1
1	2
2	3
3	4

Event loop — handles asynchronous operations by executing callback in non-blocking manner.

Rest



Collects all arguments

allows a function to take an indefinite number of arguments and bundle them in an array.

```
function sum(...args) {
    return args.reduce((add, el) => add + el);
}
```

RANKA
DATE / /
PAGE

Rest means arguments bhar do fn me

arguments
predefined collection

```
function min() {
    console.log(arguments);
}
```

```
function sum(...args) {
    return args.reduce((sum, el) => sum + el);
}
sum(1, 2, 3, 4, 5, 6)
↳ 21
```

Destructuring →

```
let names = ["tony", "bruce", "peter", "steve", "abc", "xyz"]
// let winner = names[0];
// let secondpos = names[1];
or
let [winner, runner, ...others] = names;
```

Destructuring objects:

```
const student = {
    name: "Karan",
    age: 14,
    username: "K123",
    password: 1234,
    subjects: ["hindi", "math"]
};

const { username, password } = student;
```

instead of
`let username = student.username`
`let password = student.password`

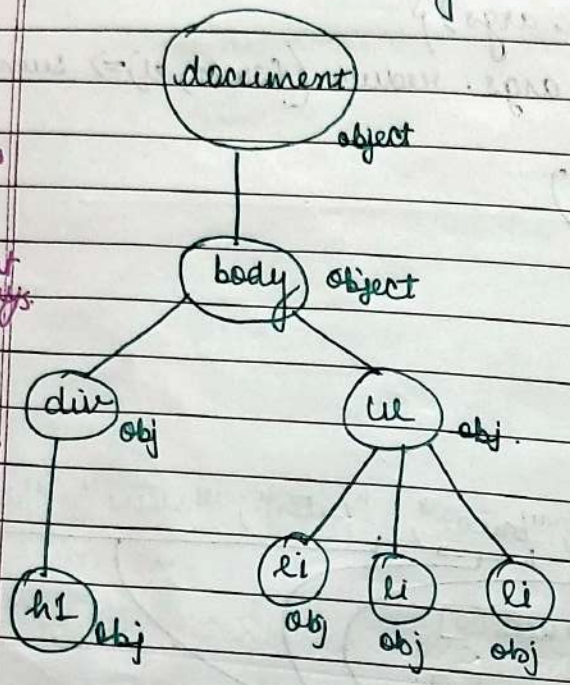
```
const { username: user, password: pass, city = "pune" } = student;
user → karan
pass → K123
city → pune
```

DOM (Document Object Model)

The DOM represents a document with a logical tree

It allows us to manipulate / change webpage content (HTML)

every element of HTML becomes an object in javascript




```
<body>
  <div>
    <h1> Todo </h1>
  </div>
  <ul>
    <li> Eat </li>
    <li> Code </li>
    <li> Sleep </li>
  </ul>
</body>
```

console.dir(document)

{ ... }

every info of the html document stored in this object (parent object).

Key:value.

console.dir(document);

"document {

location: ____

activeElement: ____

all: HTMLAllCollection(39) [html, head, meta, 0: 1: 6: link, 7: body... 9:

}

all: HTMLAllCollection (36) — no. of tags

[0: html, 1: head, 2: meta ... 8: h1]

key → value

array like each tag is given an index.

Changing object values from javascript:

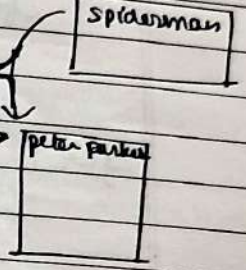
to access value of tags:

```
console.dir(document.all[8].innerText);
```

spiderman

→ o/p → Spiderman.

```
document.all[8].innerText = "Peter Parker";
```



selecting Elements:

getElementById → Returns the element as an object or null (if not found)

```
document.getElementById(id);
```

```
document.getElementById("mainImg");
```

returns <img src="spiderman_img.png" id="mainImg"> : Object

```
document.getElementById("mainImg").id;
```

→ mainImg

GetElementsByClassName:

```
document.getElementsByClassName("h1id");
```

→ h1 is class name

→ empty HTMLCollection[] returned — length = 0

```
console.dir(document.getElementsByClassName("boxlink"));
```

→ HTMLCollection(4)
- 0: a.boxlink
- 1: a.boxlink
- 2: a.boxlink
- 3: a.boxlink

An HTML Collection is returned

Indexes

a → anchor tag. boxlink → boxlink class.

length: 4

```
let smallImages = document.getElementsByClassName("oldImg");
for(let i=0; i<smallImages.length; i++){
    console.dir(smallImages[i]);
}
```

```
for(i=0 to --){
    smallImages[i].src = "spiderman.png";
    console.log("value of image ${i} changed");
}
```

getElementsByTagName → Returns the Elements as an HTML Collection or empty collection (if not found).

```
document.getElementsByTagName("p")
```

→ HTMLCollection(2)
- 0: p
- 1: p#id

→ `document.getElementsByTagName("p")[2].innerText;`

select only single element → if multiple with same exist then they take 1st one element

QuerySelector → Allows to use any CSS selector.

```
document.querySelector("p");
document.querySelector("  ")
```

→ similar to CSS Selectors
- .classname
- #Idofelement
- tag

div a

document.querySelectorAll("div a"); → returns all div a

→ returns a Nodelist (p)
length
0: a.boxlink
1: a.boxlink
2: a.boxlink
3: a.boxlink,

all child a's of div not just the firstone.

Manipulation of HTML elements using JS.

Using properties & Methods →

innerText : shows the visible text contained in a node.

textContent : Shows all the full text (hidden also shown) (similar to how written in html file).

innerHTML : Shows the full markup (bold tags, anchor tags etc... also shown)

```
let para = document.querySelector('p');
para.innerText;  → what visible on browsers scrn.
para.innerHTML;  → also displays anchor tags, etc...
para.textContent;  → similar to what we have written in actual HTML file.
```

recognises tags

```
para.innerText = "Hi I am PeterParker";
para.innerHTML = "Hi I am <b> PeterParker </b>";
```

Manipulating attributes:

getters / setters
```
obj.getAttribute(attribt);
obj.setAttribute("attribute", value);
```

```
img.setAttribute('src', 'spiderman.jpg');
```

img.getAttribute ("src")
→ spiderman.jpg

Manipulating Style:

CSS: background-color
JS: → Camel case → backgroundColor

obj.style

let head = document.[illegible]
= document.querySelector ('h1');

head.style;

head.style.color = 'Green';

obj.style
→ can only used for inline style.
→ has no access to and no effect on CSS external file.

MANIPULATING STYLE: using classList

obj.classList

- classList.add() to add new classes
- classList.remove () to remove classes
- classList.contains() to check if class exists.
- classList.toggle () to toggle b/w add & remove.
  → जो है उसका उल्टा करके दे देगा

html
→

JS → img = document.querySelector('img')
→ img.classList;
0: "abc"
1: "xyz"
length: 2
value: "abc xyz"

html → `<img src="_" >`

JS → `img.classList.add("abc");`

→ `<img src="_" class="abc" >` ... all styles of class abc applied

css →
```
.abc {
    background-color: red;
}
```

obj.~~setAttribute~~('class', 'abc');

setAttribute not preferred for this purpose / anyone class left / all other class removed

Navigation :

obj.parentElement

obj.children

obj.previousElementSibling

obj.nextElementSibling

```
<div class="box">
  <h4> </h4>
  <ul>
    <li>
    <li>
    <li>
  </ul>
</div>
```

```
let h4 = document.querySelector('h4');
h4.parentElement
o/p → <div class="box">...</div>
h4.children
→ HTMLCollection [0]
box.children;
o/p → HTMLCollection [2] [h4, ul]
box.childElementCount;
o/p → 2
li.previousElementSibling;
→ <li>
li.nextElementSibling;
→ <li>
```

Adding Elements :

document.createElement('p');

create then insert into document element.

document.createElement('p'); ⟶

let newP = document.createElement('p');
newP.innerText = "Hi! I am here";

let body = document.querySelector('body');
body.appendChild(newP);

<body> Hi! I am here

let btn = document.createElement('button');
Console.dir('btn');
btn.

btn.innerText = "click me!";
box.appendChild(btn);

newP.append("this is appended text");

newp ⟶ hi! I am here this is appended text.

obj. InsertAdjecentElement(position, element)

- beforebegin
- afterbegin
- beforeend
- afterend

obj. appendChild(element)

obj.append(element) ⟶ at last

obj. prepend(element) ⟶ at starting

RANKA
DATE / /
PAGE

Removing Elements

removeChild (element)

remove (element)

body.removeChild(btn);

btn.remove()

DOM EVENTS

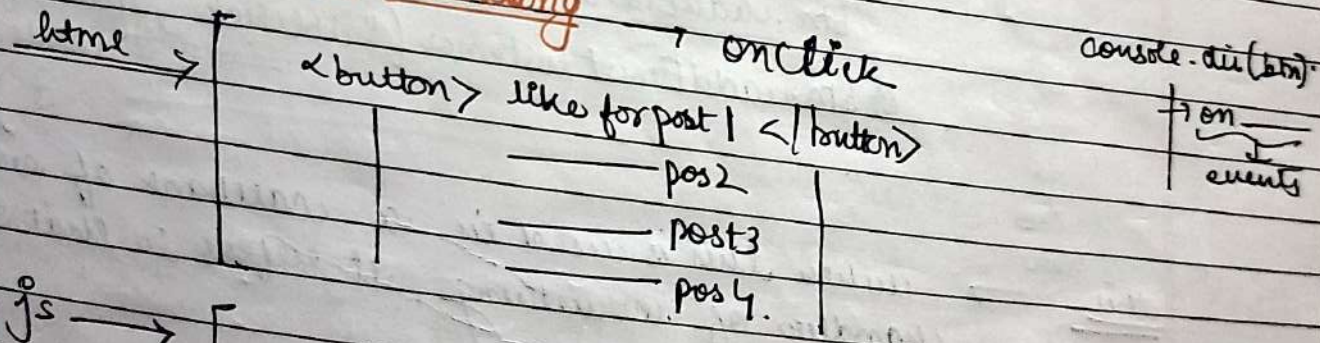
Events → signals that something has occured. (user inputs / actions)

inline → action on click.

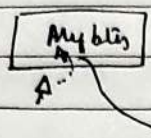
```
<button onclick = "console.log("Button clicked");" >
Click me
</button>
```

it is not advisable to use inline event handling generally.

DOM events and handling



```
btn.onmouseenter = function () {
    console.log ("entered into btn");}
```



similar to hover over the button.

Event Listener:

addEventListener:

element.addEventlistener (event, callback)

```
btn.addEventlistner ("click", function () { console.log
    ("button clicked"); });
```

```
for (btn of btns)
{ btn.addEventlistener ("click", sayHello);
  btn.addEventlistener ("click", sayname);
  btn.addEventlistener ("dblclick", function() {
      console.log("Hi")
  });
```

event — function to be performed

this → when this is used in a callback of event handler of something, it refers to that something

```
let btn = document.querySelector ("button");
btn.addEventlistener ("click", function () {
    console.log(this); });
```

this → element/ variable on which addEventlist is called

button.

function changeColor() {
    console.dir(this.innerText);
    this.style.backgroundColor = "blue"; }
btn.addEventListener("click", changeColor);
p.addEventListener("click", changeColor);
h1.addEventListener("click", changeColor);

Keyboard Events →

let btn = document.querySelector("button");
btn.addEventListener("click", function(event) {
    console.log(event);
    console.log("button clicked");
});

default argument

o/p → [PointerEvent { ... pointerType: "mouse", ... }
button clicked]

btn.addEventListener("dblclick", function(event) {
    console.log(event);
    console.log("btn clicked"); });

o/p → MouseEvent { ... }
btn clicked

<input> [clickme]

let inp = document.querySelector("input");
inp.addEventListner("type", listner);

options / use capture

Keyboard events

- Keydown → fired when key is pressed
- Keypress → fired when a key that produces a character value is pressed down
- Keyup → fired when key is released

```
inp.addEventListener("Keydown", function(event){
    console.log(event);
    console.log("Key was pressed");
});
```

Keyboard events

event

- code → 'Space', 'KeyA', 'Semicolon', KeyA, KeyB, KeyC, Enter, Slash
- Key → 'Shown on screen' → a, b, c, Enter, /

```
input.addEventlistner("Keydown", function(event){ console.log(event.key);
    console.log(event.code);
    console.log("Key pressed");
}
```

o/p: a ——→ event.key
KeyA ——→ event.code
Keypressed

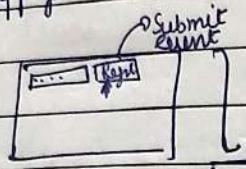
```
input.addEventlistener("keydown", function(event){ console.log("Code=", event.code);
if(event.code == "ArrowUp"){
    console.log("character moves up");
else if(event.code == "ArrowDown")
    { console.log("Character moves back");}}
;
});
```

Form Events →

```
<form action = "/action" >
    <input placeholder = "type here" >
    <button> Register </button>
```

app.js →

```
let form = document.querySelector ("form");
form.addEventListener ("submit", function (){
    alert ("form submitted");});
```



alert form submitted. → /action

if we not → event.preventDefault();

↳ redirection will not be done. and just alert is shown.

Extracting form data →

```
let form = document.querySelector ("form");
form.addEventListener ("submit", function (event) {
    event.preventDefault();
    let inp = document.querySelector("input");
    console.dir (inp);
    console.log (inp.value);
});
```

form.elements
↳ HTML Form Controls Collection (3)
[input #user] → form.elements[0]
input #pass → form.elements[1]
button → form.elements[2]

```
form.addEventListen ("submit", function(e
; let user = document.queryselector('#us
console.log (user.value);
alert (`hi ${user.value}`); });
```

```
form.addEventListener("submit", function(event){
    event.preventDefault();
    console.dir(form);
    let user = this.elements[0];
    let pass = this.elements[1];
    console.log(user.value);
    console.log(pass.value);
});
```

→ HTMLFormControlsCollection
[user → elements[0]
pass
buttons

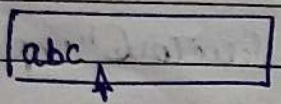
(major data changes)

Change event: The change event occurs when the value of an element has been changed. (only works on <input>, <textarea> and <select> elements).

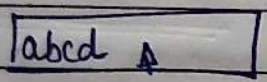
Input events → fires when the value of an <input>, <select> or <textarea> element has been changed even a little bit.
(minor changes)

change →

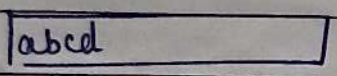
```
user.addEventListener("change", function(){
    console.log("input changed");
    console.log("final value", this.value);});
```



o/p → loading



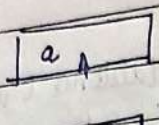
o/p → nothing



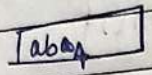
o/p → input changed
final value → abcd.

Anchored outside container

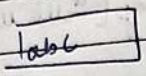
```
user.addEventListener("input", function(){
    console.log("ip changed");
    console.log("finalvalue", this.value);
});
```



input change
finalval : a



input ch
finalval: ab



input ch
finalvalue = abc;

anycharactr keys trggr input event, othrskeys, like ↑ ↓ donot

```
let input = document.querySelector("input");
let h2 = document.querySelector("h2");
input.addEventListener("input", function(){
    let filteredinput = input.value.replace(/[^a-zA-Z\s]/g, "");
    h2.textContent = filteredinput; });
```

replace(/[^a-zA-Z\s]/g, '')

replace anything with empty string if it is not a-z A-Z or a space (\s)

Metacharacters

^ → Beginning of a line

[^] → negation

$ - end of line

\s — white space

g → global flag that tells regexp to replace all occurences not just the first.

Reg exp is enclosed with forward slashes / /

allowing only 0-9 → replace(/[^0-9]/g, '')

allowing only alphanumeric → (/[^a-zA-Z0-9]/g, '')

RANKA
DATE / /
PAGE

Event Bubbling →

```
<div>
  <ul>
    <li> one </li>
    <li> two </li>
  </ul>
</div>
```

```
div.addEventListener("click",
    function() {
    console.log("div was clicked")
    });
```

```
ul.addEventListener("click",
    function() {
    console.log("ul was clicked")
    });
```

```
for(li of list){
  li.addEventListener('click',
  function(){ console.log
    ("li was clicked")
  });
}
```

o/p I

click div → div was clicked

click ul → ul was clicked
div was clicked

click li → li was clicked
ul was clicked
div was clicked

Event Bubbling

↓

to prevent it

```
ul.addEventListener("click", function(event){
    event.stopPropagation();
    console.log("ul was clicked");
});
```

Event delegation → The properties or EventListners added to parent also work for children.

to get the child who was clicked

`event.target.nodeName` = element clicked & triggered action.

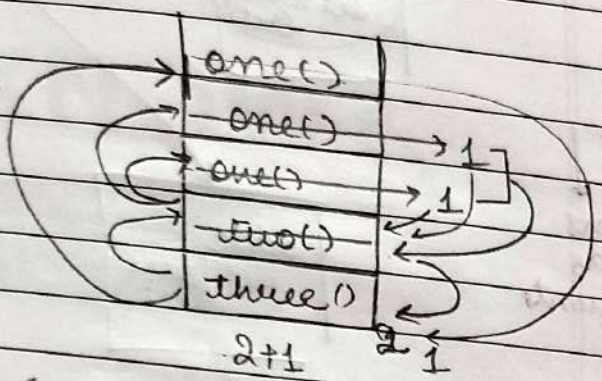
open new browser window in javascript:

window.open(URL, windowname, [windowfeatures]);

8:15:00

CALL STACK →

```
function one()
{ return 1; }
function two(){
   return one() + one();}
function three(){
   let ans = two() + one();
   console.log(ans);}
three();
```



Break points : used to visualize call stack and for debugging purpose

on browser → dev tools → Sources → app.js.
Add break points

JavaScript is Single Threaded → एक Time पर एक ही काम कर सकती है JavaScript.

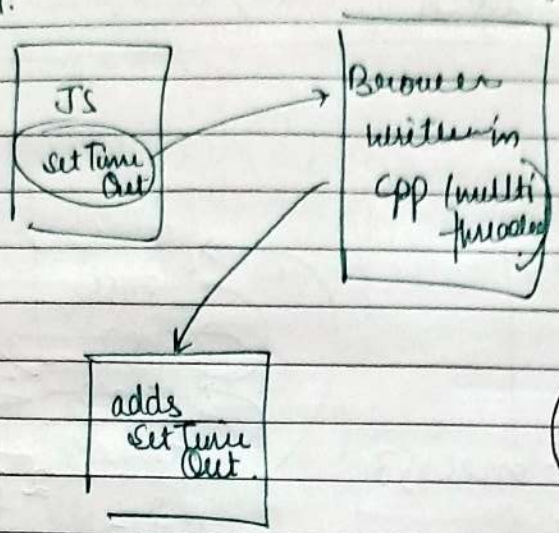
Callbacks, setTimeOut

```
setTimeOut(() => {
    console.log("apna college");}, 2000);
setTimeOut(() => {
    console.log("hello world");}
    , 2000);
console.log("hello");
```

op →
hello
apna college
hello world

since js is single threaded how did it run these simultaneously

it happens as it is not done by js, browser does this.



Callback → Functions passed as arguments

Javascript is of synchronous nature

but can be done asynchronously with the help of ~~set timeout~~ setTimeout callback.

Callback Hell →

```
function changeColor(color, delay, nextColorChange)
{ setTimeOut(() => {
    h1.style.color = color;
    if (nextColorChange) nextColorChange();
}, delay); }

changeColor("red", 1000, () => {
    changeColor("orange", 1000, () => {
        changeColor("green", 1000, () => {
            changeColor("yellow", 1000, () => {
                changeColor("blue", 1000)
            });
        });
    });
});
```

multiple callbacks.

callback-nesting → callback hell.

To deal with Callback hell → we use promises, async, await

```
saveToDB(
  "hello", () => {
    console.log("success")
    saveToDB( "hello2", () => { console.log(success2)
      saveToDB( "hi", () => { console.log("success3:
      }, () => {
        console.log("failure3 : weak connection")
      })
    }, () => {
      console.log("failure2 : weak connection")
    })
  }, () => {
    console.log("failure1 : weak connection")
  })
```

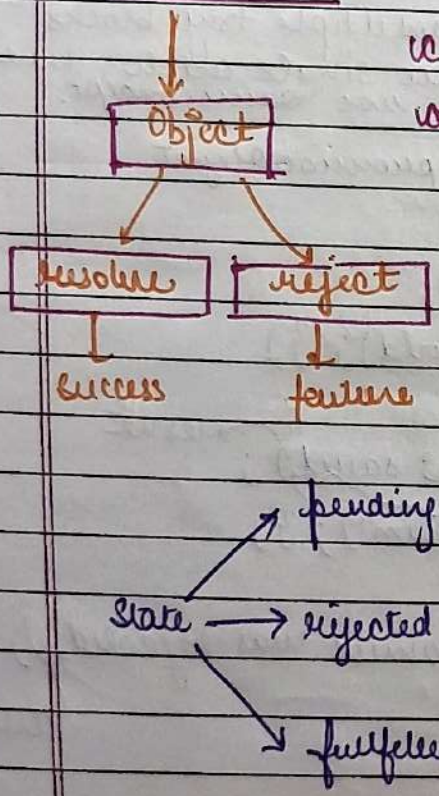
Nested callbacks

→ simple meaning

if success
→ do database operation & next call

if failure
→ donot call next operation.

PROMISES → The promise object represents the eventu completion (or failure) of an asynchronous operat and its resulting value.



function (asynchronous)

```
function Save(data) {
  return new Promise( (resolve, reject)
  => { let internetSpeed = Math.floor
       (Math.random()*10)+1;
    if (internetSpeed > 4) {
      resolve("data saved");}
    else { reject("weak connection")
    }})}

Save("Hi I am Here")
```

RANKA
DATE / /
PAGE

then() and catch()

function save(____

____); → simila to previous

Now →

let request = save ("hi i am here");
request.then(() => {
console.log ("promise fullfilled");
console.log (request);
})
.catch (() => {
console.log ("promise rejected");
console.log (request); });

then() → success of promise then do this

catch() → failure ? catch exception.

Promise chaining →

like multiple try blocks can have single catch we use same concept.

saveToDB("apna college") ----→ promise object

.then(() => {
console.log ("data 1 saved");
return saveToDb("hello world"); })

result ("success")

.then((result) => {
console.log ("data 2 saved");
return saveToDB("Ram"); })

result

error ("failure")

.catch(() => { console.log ("promise was rejected"); });

error

More compact & readable.

30 day
5 to 30.

RANKA
DATE / /
PAGE

async keyword

→ return a promise by default

```
async function agreet(){
    return "hello world";}
```
→ returns a promise.

let hello = async () => { }; returns a promise

function

normal → P: fulfilled + return

error → P: rejected

```
agreet()
  .then((result) => {
      console.log("result:", result);
  })
  .catch((err) => {
      console.log("promise rejected");
  });
```

```
async function kii(){
throw "error 404";
return "hello";}
```

Await → pauses the execution of its surrounding async function until the promise is settled (resolved or rejected)

await can only be used inside async functions

```
function getNum(){
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            let num = Math.floor(Math.random()*10)+1;
            console.log(num); resolve();}
        ,1000);});}
```

RANKA
DATE / /
PAGE

```
async function demo(){
    await getNum();
    await getNum();
    await getNum();
    getNum();
}
```

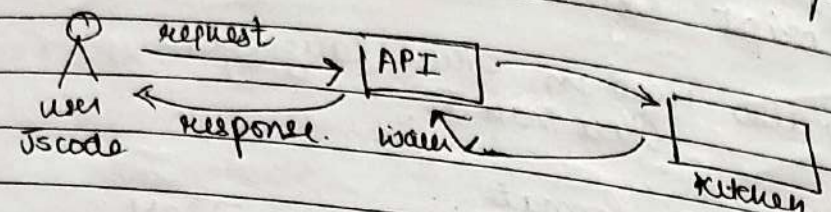
0 → after 1s
5 → after 1s.
6 → after 1s
7

```
function changecolor(color, delay){
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            h1.style.color = color;
            resolve("color changed");
        }, delay);
    });
}

async function demo()
{
    await changecolor("red", 1000);
    await changecolor("blue", 2000);
    await changecolor("green", 1000);
}
```

```
if(---){
    reject("---");}
```

```
try{
    await chang(---);
    await ---->
catch(e){
    ---
}.
```

API (application programming interface)



amazon.in → request → amazon server → response → HTML + CSS + JS → webpage in browser.

S/w ← API → S/w
interact

API that use http → web APIs

User client → req → API → link (endpoint) → APIs → data (JSON)

twitter API → bot accounts
automatic replies & tweets
without even opening app.

Key → password used for using some APIs.

free API → catfact.ninja/fact.
{"fact": - - - - - - - - }
→ data is returned

json data format
↓
made for computers to read.

API → dog.ceo/breeds/random
↓
returns random
{"message": "bot - - - ", "status", "success"}.

Google maps API → directions API → modes, factors } documentation.

JSON

→ JavaScript Object Notation

www.json.org

data format → also used in other languages other than JS.

Xml → extensible data format

→ earlier used as data format

→ similar to html. custom tags.

JSON

→ Key value pairs

in js not passed as a string

→ name: "Kamal". not a string

JSON object values cannot be undefined like js objects.

Values of objects allowed: object, array, string, number, "true", "false", "null".

JSON

→ "name": "Kamal" is a string.

check json valid format json documentation.

JSON —response→ String format in Javascript

→ string formatted json data

JSON.parse(data)

→ to parse a string data into a JS object.

let jsondata = '{"fact": "He loves you"}'; string.

let validresponse = JSON.parse(jsonres);
console.log(validResponse.fact);

op → He loves you.